Laravel, Git, Async, Minimal APIs, CSVHelper, JQuery

CODE

JAN
FEB
2022

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 8.95 Can $ 11.95

CODE

Kafka
Event-Streaming Platform

Managing CSV Files With CSVHelper

Exploring Laravel: Part 2

Going Knative on Docker Containers

01
7 25274 56130 8

Microsoft Azure + AI Conference
CO-PRODUCED BY
Microsoft & DEVintersection

DEVintersection Conference
POWERED BY
Microsoft & DEVintersection

The Intersection of Technology

SCOTT GUTHRIE
Executive Vice President, Cloud + AI Platform, Microsoft

CHARLES LAMANNA
Corporate Vice President, Business Applications & Platform, Microsoft

SCOTT HUNTER
Vice President Director, Product Management, Azure, Microsoft

SCOTT HANSELMAN
Partner Program Manager, Microsoft

JEFF FRITZ
Senior Program Manager, Microsoft

KATHLEEN DOLLARD
Principal Program Manager, Microsoft

JOHN PAPA
Principal Developer Advocate Lead, Microsoft

KARUANA GATIMU
Principal Manager, Customer Advocacy Group, Microsoft Teams Engineering, Microsoft

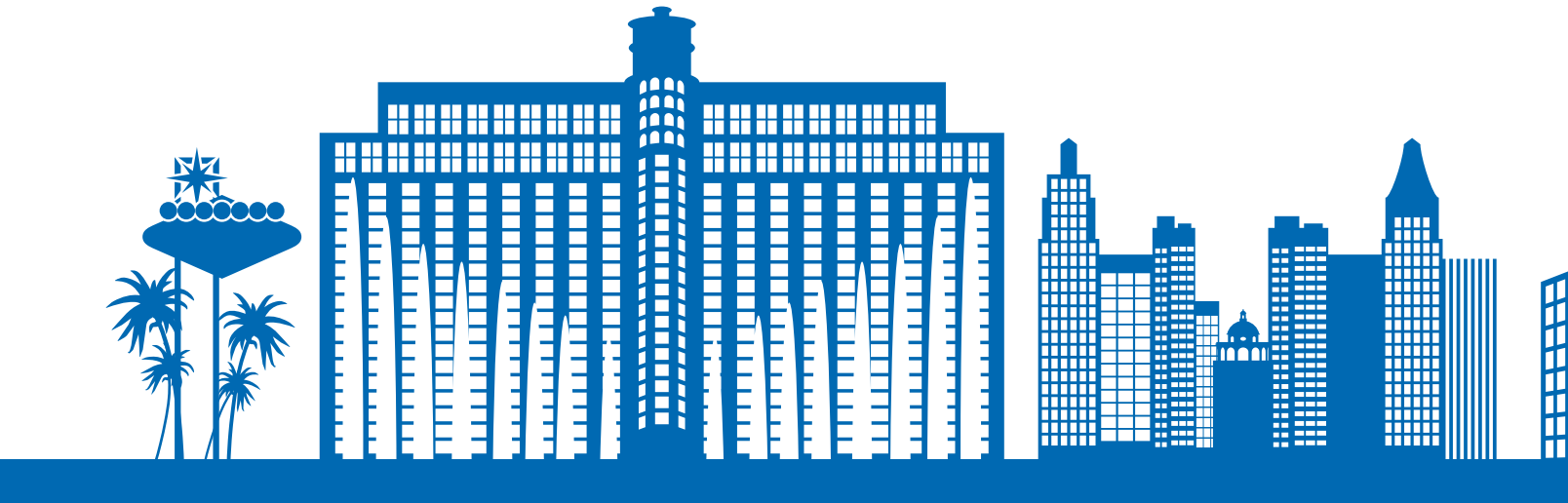
DAN WAHLIN
Cloud Developer Advocate Manager, Microsoft

JEFF TEPER
Corporate Vice President, Microsoft Teams, Microsoft SharePoint, Microsoft OneDrive, Microsoft

DAN HOLME
Principal Product Manager Lead for Yammer, Microsoft

PAUL YUKNEWICZ
Lead Product Manager, Microsoft

and many more!



SQL Server & Azure SQL Conference
POWERED BY
Microsoft & NextGen

Microsoft 365 COLLABORATION CONFERENCE
Microsoft Teams
Microsoft SharePoint
Microsoft Power Platform

Featuring a bonus track dedicated to Microsoft Viva

April 5–7, 2022 Las Vegas, NV

Workshops April 3, 4, 8 **MGM GRAND**

Here are just a few of the topics covered in sessions and workshops:

.NET 6 • .NET MAUI • BLAZOR • C# 10 • ANGULAR • AZURE • AI
AZURE SQL • MODERN DATA • SQL SERVER • KUBERNETES • DEVOPS
PROJECT DESIGN & UI • SECURITY • MACHINE LEARNING • AZURE LOGIC
MICROSOFT POWER PLATFORM • MICROSOFT TEAMS • MICROSOFT SHAREPOINT

APRIL 2022 REGISTRATION
for all conferences

When you ***REGISTER EARLY for a WORKSHOP PACKAGE,*** you'll receive a choice of hardware or hotel gift card!

Go to **DEVintersection.com** or **M365Conf.com** for details.

iPad Mini


Surface Earbuds


Surface Go 3


Surface Headphones

DEVintersection.com AzureAIConf.com M365Conf.com

203-527-4160 M–F, 12-4 EST

Features

Columns

Departments

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Bill Me option is available only for US subscriptions. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.

LEADTOOLS®

Robust Collection of SDKs Powered by AI and Machine Learning

LEADTOOLS is a collection of comprehensive toolkits to integrate recognition, document, medical, imaging, and multimedia technologies into desktop, server, tablet, and mobile solutions.

Some of our features include:

OCR/ICR	BARCODE	PDF	IMAGE PROCESSING
DICOM	PACS	MEDICAL VIEWER	VIDEO/AUDIO
FORMS PROCESSING	DOCUMENT VIEWER	DOCUMENT CONVERTER	DOCUMENT COMPARE
DOCUMENT ANALYZER	VIRTUAL PRINTER	IMAGE VIEWER	OFFICE FORMATS

Microsoft .NET · Xamarin · Windows · Java · Linux · iOS · macOS · Android · HTML5

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

Finding Inspiration

This editorial marks a huge milestone in my life: It officially marks the end of my first 20 years as editor in chief of CODE Magazine. And before you get any ideas, this is NOT my last editorial! I have many more years ahead of me to "entertain" you with my witty banter and deep knowledge of software engineering, science, music, and Dungeons & Dragons. Some of that statement is true.

When I started thinking about this editorial, I reviewed a bunch of my past editorials and was proud of what we've accomplished at CODE Magazine in the last 20 years. It's amazing how much things have changed in that time. The early 2002 issues were all about this new ".NET Initiative," Web Services, XML, and XSLT. The cloud was non-existent at the time, there was no Twitter, no Facebook—heck, Amazon's primary business was selling books. How things have changed!

As some of you know, I was hired as EIC of CODE Magazine via an instant messaging session (ICQ I think) and, to be honest, this was a dream come true. From the time I was in high school, I dreamt of being a writer. Did I want to be a tech writer? Heck no! I wanted to be a Dungeons & Dragons writer. That dream was partially filled in high school when I published my first D&D article called "The Role of Taxes." I was a geek then and I'm a true geek now. So, for those long-time readers (and new ones of course), where am I going with this? Well, I want to talk about inspiration.

As I was thinking about the things that inspired me and how to best represent them in this editorial, I decided on a picture. Like many geeks, I've spent decades collecting various geek trophies of things I enjoy. **Figure 1** shows a bookshelf containing the many, many things that provide me with comfort and inspiration. I'm going to highlight a few of them.

Star Wars

I'll never forget the opening credits of Star Wars. The title card read: "A long time ago, in a galaxy far, far away....". I fondly remember my eight-year-old self writing "books" about Star Wars with an oversized pencil. Writing new adventures for Luke, Leah, and Han were my bag.

Dungeons & Dragons

I discovered Dungeons & Dragons when I was 12 years old. I remember getting my first module on my 13th birthday called "The Glacial Rift of the Frost Giant Jarl" Yes that was the title and some 40 years later, I can still recite the names of many of these modules. The names were epic. D&D was (and still is) an amazing game and it took me to many mythological as well as real-world places. From Greek to Roman to Norse to Tolkien, every mythology was represented. As for the real-world places, I met many other gamers in high school (I was president of the Golden Dragon Club at one point) and at numerous conventions in places like Los Angeles and Milwaukie. I was deep into this game, and that inspired me to start writing about it. Like all new writers, I got a TON of rejection letters, but I persevered and had some minor successes. These successes drove me forward.

Figure 1: My collection of inspirational trophies

Programming

I was determined to be a writer until I discovered programming. Programming has always been fun for me and when I discovered databases in college, I knew what I wanted to do for a career. I started in the DOS era and have continued to write code for over 30 years now. I still find enjoyment in slinging code to this day. Over the years, I've had the opportunity to work with some great developers and have grown to be a fairly skilled programmer myself. But however skilled I became, I missed writing. It was programming skills that eventually led me back to writing.

Writing

In 1992, I decided to see if I could get published in a computer magazine. I went to a software conference and proposed an idea to Dian Schaffhauser who was an editor at Database Advisor Magazine. She accepted and I went to work writing my first article. One article led to another, and another, and another and eventually it led to writing books and finally to being EIC of CODE Magazine. I've never stopped writing. As a matter of fact, I wrote an article for this issue! Writing has been a source of inspiration to, well, keep writing. It's a sickness, I think. Talk to me about writing books some time.

Finding YOUR Inspiration

I described just a few items that inspire me. There are others: music, pop art, movies, and economics, to name a few. The trick for you is to find what inspires you and lean into it. Having sources of inspiration is what makes our world go around. I hope you find yours!

Rod Paddock
CODE

CUSTOM SOFTWARE DEVELOPMENT

STAFFING TRAINING/MENTORING SECURITY

MORE THAN JUST A MAGAZINE!

Does your development team lack skills or time to complete all your business-critical software projects? CODE Consulting has top-tier developers available with in-depth experience in .NET, web development, desktop development (WPF), Blazor, Azure, mobile apps, IoT and more.

Contact us today for a complimentary one hour tech consultation. No strings. No commitment. Just CODE.

codemag.com/code

832-717-4445 ext. 9 • info@codemag.com

The Basics of Git

I can't think of any other skill besides Git that is universally applicable to any developer. It doesn't matter if you write code in C#, JavaScript, or Python, or for Windows or Mac or really anything else, there's a solid chance that these days you use Git for source control. And no, that doesn't mean just GitHub. Git is an open standard backed by the Git community. Various other products, such as Azure DevOps, Bitbucket, and Atlassian all support Git. First things first. I'm going to avoid the lightning rod discussion of whether Git is a good product or not. The reality is that whether you like it or not, all of us use it. And let's be honest: It has proven to be scalable enough for the largest source code repositories, and it's pretty easy to get started with, too.

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.

Yet it's one of those products that really drives me mad. So I thought it might be worth writing an article, explaining the basics of Git. With a strong foundation, you can build taller buildings.

Centralized Source Control vs. Decentralized Source Control

If you've worked with older versions of source control software, such as Visual SourceSafe, Mercurial, PVCS, or many others before that, you're familiar with centralized source control. In centralized source control, there's a server in the middle that all developers talk to. Any software project is comprised of many files. If you wish to work on a certain file, you check out that file. While that file is checked out, its status is marked checked out in the centralized source control repo. If any other developer wishes to overwrite that file, they're unable to, because it's checked out to you. You need to check in your changes first, and the other developer's changes are the other developer's headache. The other developer must probably do a merge or something similar. All of this works fine, but it has two main problems.

The first issue is what happens if that central server goes down. You can continue to work on the previous snapshot you pulled from the server. But sooner or later, when you need to re-sync your changes to the server or check-in files, you hit a wall. You can't for instance, continue working with source control locally. For that matter, you can't work with an alternate remote upstream location for the meantime, such as a co-worker's source control. And what if you want source control on just your computer, without any need to share with rest of the world, for a pet project that's complex enough to deem source control?

The second issue, of course, is scale. Centralized source control repos assume a small set of developers working very closely together. These days, we all contribute to very large source control repos, which are typical in popular open-source projects. A centralized source control mechanism that relies on locking in a central location to talk with simply doesn't scale to the general complexity of large-scale repos and a disconnected working model.

Both of these issues are fuzzy in nature. Visual SourceSafe fans insist that there are workarounds to these problems. But just because you can row to Japan in a tiny boat doesn't mean it's a good idea. To the rest of the world, it's clear that we need a new approach, a decentralized source control.

The opposite of centralized source control is decentralized source control, of which Git is an example. In decentralized source control mechanism, you can have many locations with the source control repo. These locations can be servers, or they can even be your own local hard disk, or they can be a coworker's hard disk. You can merge changes between these multiple source control repos. Also, you don't rely on exclusive check-in and check-out anymore. Instead, you rely on merges and commits. This invariably has the downside of merge conflicts. Good coding patterns, good architectural practices, and writing good tests reduce this pain to some degree, although don't eliminate it.

Let's start learning Git.

Install Git

Many development tools, such as XCode, already come with Git packaged. Even if you already have Git on your computer, it's a good idea to update it. The instructions are unique per operating system. Rather than rehashing instructions here, I suggest that you visit *https://git-scm.com/book/en/v2/Getting-Started-Installing-Git* and follow the instructions per your operating system and install Git on your computer.

Once you've installed it, you should be able to run the command "git" on terminal. For Windows, you'll notice that after installation, you get a special terminal called "Git bash". This is a special terminal/command window on Windows that tries to emulate a Unix-like terminal. You're welcome to use it, although I've also used Git through the PowerShell window and never run into any issues. I do feel that you should lean on a Unix-like terminal even on Windows, because a lot of commands invariably end up making use of Unix-like commands intertwined with Git commands. Most devs mix and match them without even thinking about it.

Configure Git

Before you can use Git, you have to do some basic configuration. At the bare minimum, you'll need to specify a name and email—this is your information, who are you when you issue a commit. Of course, the server-side repo also authenticates you through the various means that Git supports.

You can also optionally specify a default editor and I highly recommend that you specify a line ending format as well.

Let's perform this basic configuration on your computer.

When you perform Git configuration, you can do so at one of three levels. You can do so at a global level, which affects all users on your computer. You can do so in your user profile, in which case it will affect all work on the user's profile. Or you can specify at a folder level, where you wish to have certain settings affect only certain repos. These settings go in a hidden file called ".gitconfig". My gitconfig looks like that shown in **Figure 1**.

First let's configure the username and email address. Here's how. Remember to use your own username and email address.

```
git config --global user.name "Sahil Malik"
git config --global user.email sahilmalik@winsmarts.com
```

Next, let's specify a default editor. By default, Git uses Vim. A lot of people love Vim. Personally, I never have to restart my Mac unless I'm trying to exit Vim. There are just too many damned shortcut keys to remember. No, I don't dislike it; in fact when I am ssh'ed into a Docker container, using something such as VSCode may not be an option. But I do find myself more productive in VSCode, so I'll just set that as my default editor as follows.

```
git config --global core.editor "code --wait"
```

Of course, for the above to work, VSCode should be installed and be in your path. Now, whenever you need to enter multi line commit messages, or do stuff that requires any kind of editing, VSCode pops up. Let's try this. Run the below command to edit all your settings.

```
git config --global -e
```

As you can see, VSCode pops open with your .gitconfig settings. No longer do you have to remember the shortcut "shift_ZZ" to save and exit, because this isn't Vim.

Figure 1 shows my settings that have a few additional things I haven't talked about. Your settings file may look slightly different.

Finally, let's configure end of line settings. This is a very important setting, so let's understand what this is. On Windows, an end of line looks like this:

```
text\r\n
```

On Mac/Linux, try this:

```
text\n
```

Notice the difference? Windows likes to use carriage return and new line. The reasons for this are historical, and so deeply rooted that Windows isn't going to change. But this creates a big problem when some of your developer friends are on Macs and you're on Windows. In fact, when contributing to OSS projects, this will invariably be the case. So as a best practice, perform the following configuration on Windows,

```
git config --global core.autocrlf true
```

This will cause Git to strip out the /rs (carriage returns) when checking your files in.

Getting Help

There seem to be a lot of commands you need to remember here. Luckily, there's help. At any point, you can issue the following command to get help:

```
git --help
```

And if you wish to get specific help on a sub command, you can issue a command like this:

```
git config --help
```

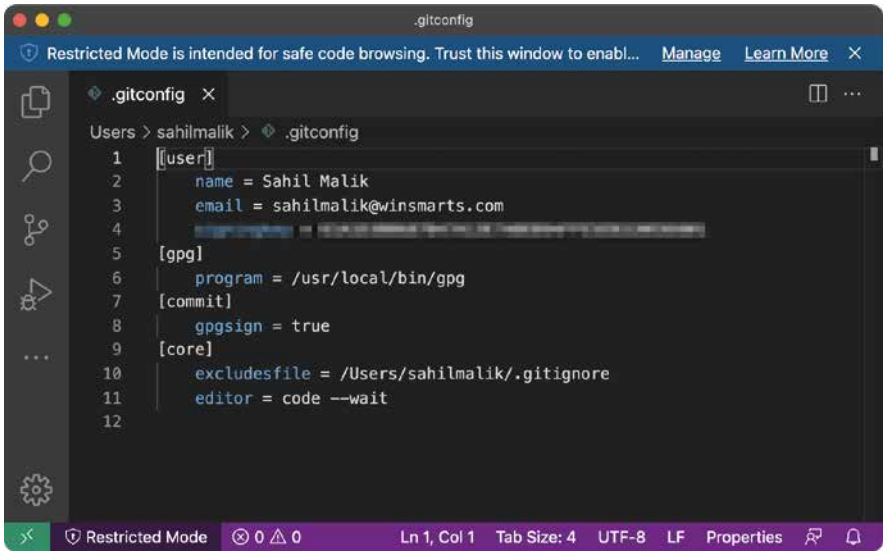


Figure 1: VSCode as my default editor for Git.

Figure 2: Git with tab completion

One other thing I highly recommend is that if you're on a Mac or Linux environment, set up zsh with a theme called "oh-my-zsh". It makes great use of the Git plug-in and gives you syntax highlighting on terminal and even tab completion. This can be seen in **Figure 2**. On Windows, you can either use the instructions at *https://winsmarts.com/running-oh-my-zsh-on-windows-10-6fcb0fbc736b*, you can set up WSL2, or you can use **posh-git**.

Initialize a Git Repo

A Git repo, or repository for short, lives in a folder. Go ahead and create a new folder. I created one called "gitlearn". To initialize a new empty repository in this folder, when inside this folder in terminal, issue the following command:

```
git init
```

This creates a new Git repository in this folder. Additionally, it creates one branch in this empty repository. The name of the branch by default is "master" although they let you configure it to use "main" by default if you prefer. You can choose to change the name of the initial default branch as follows:

```
git config --global init.defaultBranch "main"
```

What makes a folder a Git repository? Inside this folder is a hidden folder called ".git". This folder is where Git likes to store all its inner workings. If you delete this folder, it's no longer a Git repository. I advise you to not hand-edit stuff inside this folder. Leave it alone.

Commit Code

Let's first understand the very basics of the Git workflow. A typical project is comprised of multiple files and folders. In

Figure 3: My prompt tells me that I have unstaged changes.

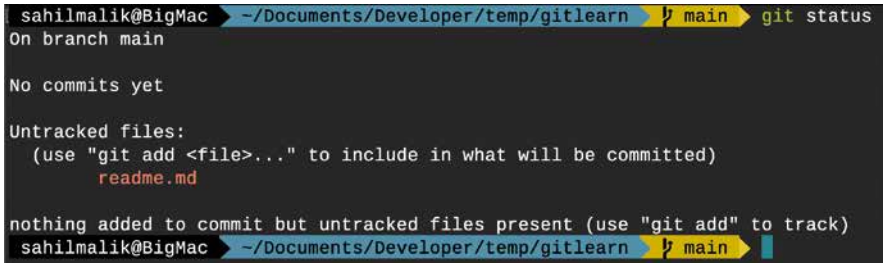


Figure 4: Git status tells me that I have an untracked file.

Figure 5: I have staged, but not yet committed.

Figure 6: My first commit

Git, you have to think in three parts: local files, committed files, and the staging area.

Your local files are your work-in-progress, also known as working copy. Here, you're making things, breaking things, and editing stuff, and edits don't preserve history. Git, of course, tracks what files are being changed, added, or deleted. But if you edit a file multiple times in a single commit, to Git, it appears as one edited file, not many versions of this edited file. This is work in progress after all.

Then you have committed files. For now, let's not mix in server and client. The cool thing about Git is that even your local hard disk has commits. I find this very useful when I'm progressively building a project, and I commit as I go along. I can revert back to a commit if I mess up or find out where I messed up using diffs etc., all without involving a server. See how productive Git is? Even without an Internet connection, I have so much power at my fingertips.

Now let's also add a server briefly. When you add committed code to your local repo, you can choose to "push" to an upstream location. That upstream location is the server. During the push, you may have to resolve conflicts, merge your team members' code, etc.

Finally, you have an area that sits in the middle of the committed area, and the working area, which is the staging area. The staging area is your "proposal to commit." Think of it as: Okay I've been working on stuff in my local area, and I'm ready to commit. And here is my proposal of what I want to commit. For instance, I propose these three files to be added, these two files to be renamed, etc. I "stage" those changes in the staging area before I commit. And then I commit.

Let's see this in action. If you've been following this article sequentially, you should have an empty Git repo. In this repo, go ahead and add a file like the next snippet. Note that my commands shown here are for the *nix shell, but you can extrapolate this on a Windows computer also.

```
echo "first file" > readme.md
```

This command creates a file called "readme.md" with "first file" as the text contents. Note how my zsh prompt has also changed in **Figure 3**. I find this color change a very convenient mechanism to know that my tree is dirty.

The yellowish color indicates that my repo has unstaged changes. If I wish to see what the current status of my Git repo is, I can use the following command.

```
git status
```

In my case, it should produce output as shown in **Figure 4**.

Git status tells me that I have an untracked file. That makes sense, as I just added a new file, but Git isn't tracking it for me—I haven't ever added it to my repo. To add it, I need to commit it. But before I can commit it, I need to stage it. To stage all changes, I can issue the following command.

```
git add .
```

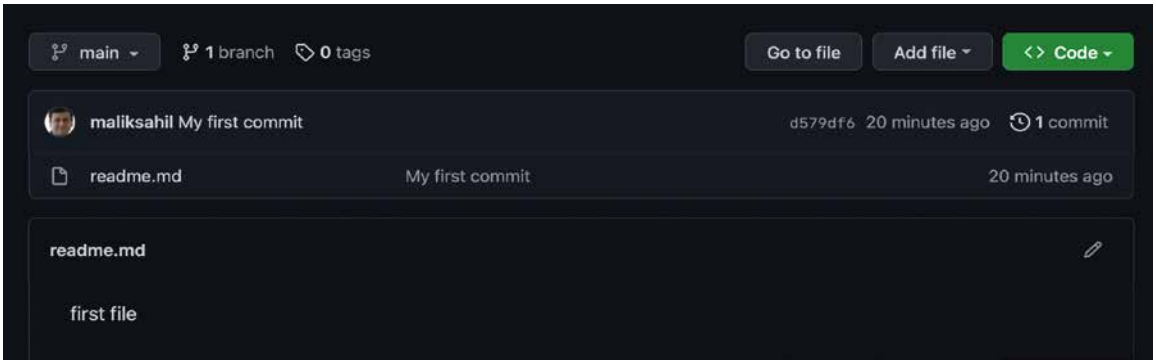


Figure 7: My code in the cloud

Or alternatively, I can say **git add** and pass in the specific files I wish to stage. Notice again, in **Figure 5**, how my prompt has changed.

Now to commit, I simply issue the following command.

```
git commit
```

This should pop open your default editor, in my case VSCode, to enter a commit message. I could also say **Git commit -m "message"** to avoid opening the editor. I'll enter some message like "My first commit" and save to commit. Now my prompt should change as shown in **Figure 6**.

Congratulations, you just did your first commit. Now try executing Git status again. It should tell you that your working tree is clean. You can see your history of commits by executing the following command.

```
git log
```

At any point, I really encourage you to type **-h** in front of any command and examine what other options are supported.

That was fun, but there's still so much more to learn: branches, server-based stuff, forking, merging. So stay the course, young Padawan.

Push Code to a Server

You're a serious developer. You aren't just writing this to learn Git: You want to do some serious work. That means that you need a server-based Git repo that can scale to the Internet. An easy way to get such a repo, for free, is github.com. Feel free to use any other product you wish. For my purposes, I created a repo at *https://github.com/maliksahil/gitlearn*. (This is a private repo, so you won't be able to access it. All of these repos are protected by permissions, so you won't be able to commit to mine, even if it were a public repo. You should go and create your own.)

Now back to my local Git repo. I wish to push my local code into the server-side repo. How does my local repo know where to push to? The answer is that I need to add a remote origin, and here's how you do it:

```
git remote add origin
 git@github.com:maliksahil/gitlearn.git
```

Now wait a second. Let's unpack this a bit. What's that funny looking syntax? How did Git authenticate me?

First of all, Git remote lets me manage tracked remote repositories. By saying Git remote add, I'm saying that I wish to add a remote tracked repository. The origin keyword indicates that here is where this project was originally cloned from. You didn't clone your project from the remote repository, but you're basically saying that in the process of setting things up, in future, developers can clone from here. The final parameter is the URL.

The way authentication works here is that by default, GitHub uses **username password**. But that's neither secure nor manageable. So it also supports ssh, which is what I have set up on my computer. Finally, you can use credential helpers to use alternate mechanisms of authentication as well.

dtSearch®

Instantly Search Terabytes

dtSearch's **document filters** support:
- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:
- efficient multithreaded search
- easy multicolor hit-highlighting
- forensics options like credit card search

Developers:
- SDKs for Windows, Linux, macOS
- Cross-platform APIs cover C++, Java and recent .NET (through .NET 6)
- FAQs on faceted search, granular data classification, Azure, AWS and more

Visit dtSearch.com for
- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

dtSearch.com 1-800-IT-FINDS

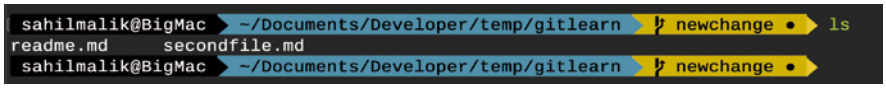


Figure 8: Two changes are ready to go.

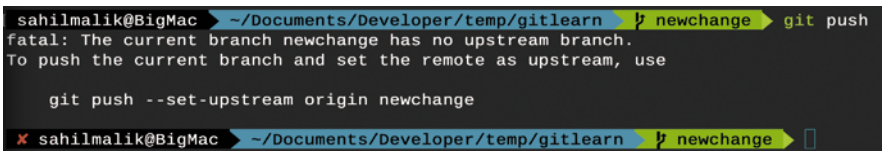


Figure 9: I have no upstream branch.

Figure 10: I have branches.

Figure 11: Merging branches

SPONSORED SIDEBAR:

Ready to Modernize a Legacy App?

Need FREE advice on migrating yesterday's legacy applications to today's modern platforms? Get answers by taking advantage of CODE Consulting's years of experience by contacting us today to schedule your free hour of CODE consulting call. No strings. No commitment. Nothing to buy. For more information, visit www.codemag.com/consulting or email us at info@codemag.com.

Go ahead and execute the Git remote add origin command, as shown in the last snippet.

Next, you need to set the upstream branch. I haven't yet had a chance to talk about branches, but since you have only branch "main", that will be your upstream branch. The idea is that an upstream branch is what's tracked on the remote repository by your local branch. My local "main" needs to mirror the server side "main", so my "main" branch is a great choice for an upstream branch. To set the upstream, and to push my "main" into "origin", I use the following command:

```
git push -u origin main
```

Now visit the GitHub repo in your browser and your code should be visible, as shown in **Figure 7.**

Modify Code

Okay, at this point, you should have a server-side repo at *https://github.com/maliksahil/gitlearn* and a locally cloned repo. Assuming that you don't have a locally cloned repo, you can use "git clone" to clone the repo from the server location. You know you can pass the **--help** parameter to any command, right? Try doing a **git clone** yourself.

Now I wish to make changes. A typical software project contains a number of files. You also go through many releases. Although nothing stops you from making changes to "main", it's generally considered a bad idea. Most real-world repositories set a policy on the Git repo to prevent making changes to the "main" branch. The idea is that you create an issue. Then you discuss what you wish to do on that issue. You associate the issue with the files you're changing and you create a separate branch for your changes.

Aha! You're into branches now. What is a branch? For now, just think of it as a copy you've made of your code. I'll get into this in a minute.

You create a branch and you make your changes there. And then you "merge" your changes into "main" via a pull request.

Gosh that's a mouthful. Before I get any more confused, let's see this in action.

First, in my local repo, let's create a branch.

```
git branch newchange
```

This command has now effectively given you a copy of "main". Don't worry, it isn't literally a full copy. Git is smart enough to abstract the details only for changes. For you, it feels like a copy. Before you can start working on this copy, you need to check out this branch, as follows:

```
git checkout newchange
```

I could have also abbreviated the above two commands into one, effectively saying "create a branch and check it out" like this:

```
git checkout -b newchange
```

Alternatively, I could create a branch in the server-side repo and pull the changes using Git pull etc. That would be very useful if your coworkers have created a branch that they've pushed to the server and you wish to work on it collaboratively.

At any point, you can run "git branch" to verify which branch you're on and which branches are available locally.

Now that you've checked out the newchange branch, let's make some changes. Modify the first file by appending some text.

```
echo "more changes" >> readme.md
```

And create a new file.

```
echo "a new file" > secondfile.md
```

Now, you should have to changes ready to go, as can be seen in **Figure 8**.

Now let's stage these changes, commit them locally, and then push them into the cloud.

First, stage:

```
git add .
```

Then commit:

```
git commit -m "My second commit"
```

Look at you, issuing Git commands like a pro. I'm so proud. Now let's push it to the cloud.

```
git push
```

The last command didn't work. You should see an error, as shown in **Figure 9**.

This makes sense if you think about it. I never told my Git repo which upstream location "newchange" should be sent to. And it gives me a helpful command to fix it. So go ahead and run that command, which then sets the upstream location and pushes my changes.

```
git push -u origin newchange
```

Oh yes: **-u** is a shorthand for **--set-upstream**

This is where the fun starts. Observe **Figure 10**.

See, in **Figure 10,** I effectively now have multiple versions of my code base. Isn't this great? I can now revert back to a production version in "main" while switching to a dev version in "newchange".

This brings up a question. How do I get my changes from "newchange" into "main"? There are two ways.

First, you can do a pull request. This means, you go to the Git repo and issue a PR (short for pull request). This is you asking, hey, I would like to merge these changes into main, and usually you'd also have some reviewer on the PR. The idea is that you don't have permissions to merge into main, or, as a policy, you wish to have an extra set of eyes look at your code. It's possible to set these policies on your repo, and most real-world projects have such policies.

The other mechanism is that you can merge from newchange into main and then push main to an upstream location. This is where, typically, you have both branches under your control. For instance, perhaps you have created a branch of a branch, but both branches are your dev work.

```
git merge newchange main
```

This workflow can be seen in **Figure 11**.

You can push this and your remote repo will reflect these changes. But I have other plans. Let's use the PR method. Visit your GitHub repo, and you should now see a nice helpful message, as shown in **Figure 12**.

Use that "Compare & pull request" button to create a PR. This gives you a nice overview of the changes, the comments, files committed, approvers, labels, etc., which is great for a development workflow. You can also set up bots to do some basic review for you, and all sorts of other automation involving humans. When you're done, you can merge the pull request, and delete the branch, as shown in **Figure 13**.

Now you've merged the PR, deleted the branch, and your changes are in main. You can feel free to also delete your local "newchange" branch as follows:

```
git branch -d newchange
```

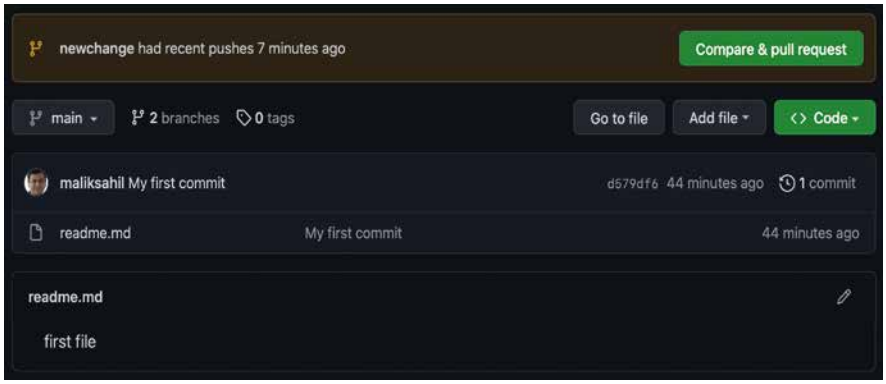


Figure 12: Creating a PR

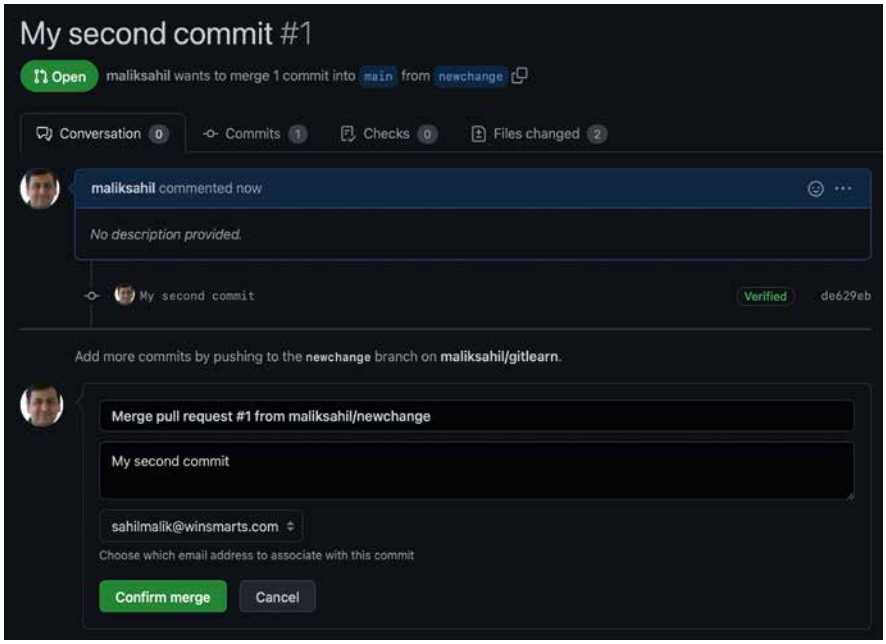


Figure 13: Merge a PR

Move, Rename, or Delete Files

I'll keep this section short because Git automates this nicely. And to save time and ink, I'll do everything in the "main" branch. Go ahead and perform the following changes to your repo.

```
mkdir afolder
mv secondfile.md afolder
mv readme.md dontreadme.md
```

You created a new folder and moved the secondfile.md into that folder, effectively deleting it from the root folder and adding a new file in afolder. Then you renamed readme.md to dontreadme.md.

Now, running a Git status basically tells me everything that I just did. You can see this in **Figure 14**.

And now I can stage, commit, and push my changes as follows.

```
git add .
git commit -m "More changes"
git push
```

You just made some amazing changes and pushed them to the remote repo. Best of all, you did so using the concepts you have learned so far. I encourage you to repeat these changes using a branch and do a pull request to solidify your knowledge.

Ignore Files

In any development project, you'll have files that you don't wish to check-in as a part of your source code. These may be node_modules in a node project, or bin, obj folders in a .NET project. Your dev tools or development environment need these files, but they're downloaded or generated on the fly. Sometimes they're even specific to the operating system you're working on. Or perhaps you have configuration files with secrets or keys specific to the developer's environment. There are many situations where you want certain files to not be checked in.

Let's understand how you can teach Git to ignore files.

If you've been following this article, you should have a Git repo that looks like **Figure 15**.

What I wish to do now is instruct Git to ignore the "afolder" contents going forward. Also, I'm going to create another file in the root of my repo. Let's call it env.txt, and I don't want to check it in. To save time, I'll do stuff in the main branch, although in real-world scenarios, you want to branch and merge.

First, let's create the env.txt file as follows:

```
echo 'someconfig' > env.txt
```

To instruct Git to ignore the env.txt and afolder folder, I'll create a new file in the root of my repo called ".gitignore". You can also choose to create a .gitignore file per folder and have those settings apply only to that folder and its children.

In my .gitignore file, I choose to put the following text:

```
env.txt
afolder/
```

At this point, I'm going to add, commit, and push. Now let's visit my repository on github.com and examine what it looks like. This can be seen in **Figure 16**.

Are you surprised by what you see in **Figure 16**? I do see that env.txt was ignored. But why is afolder still there? It's

```
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn main • git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    readme.md
        deleted:    secondfile.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        afolder/
        dontreadme.md

no changes added to commit (use "git add" and/or "git commit -a")
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn main •
```

Figure 14: The Git status for bunch of stuff I just did.

```
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn main tree
.
├── afolder
│   └── secondfile.md
└── dontreadme.md

1 directory, 2 files
```

Figure 15: My Git repo so far

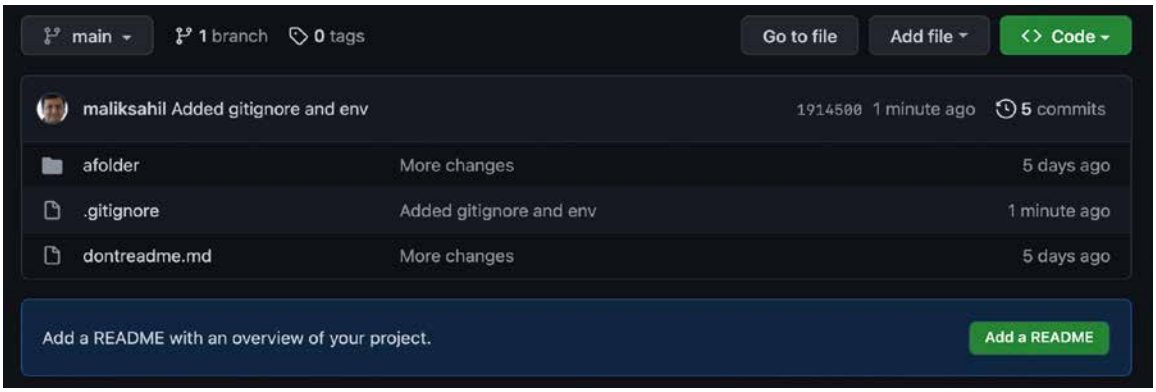


Figure 16: My Git repo with .gitignore

```
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn/afolder main echo 'sometext' > anewfile.md
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn/afolder main cd ..
```

Figure 17: The working tree remains clean even after new files are added in ignored folders.

```
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn/afolder main echo "more text" >> secondfile.md
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn/afolder main •
```

Figure 18: Making changes in an already tracked file in an ignored folder

still there because you're ignoring it going forward. This means that now if you were to put another file under afolder and try to check it in, that new file won't be checked in.

You can see this in action in **Figure 17**. Notice that the working tree remains clean, no matter what I do in afolder. Or is it?

Let's append some text in the afolder/secondfile.md file—remember that the secondfile.md file is checked into Git already. This can be seen in **Figure 18**.

Interestingly, now my working folder is no longer clean—even though I made the change in a file that resides in a folder that I've instructed to be ignored. This is because the file "secondfile.md" was already being tracked.

Why is this useful? It's useful for configuration settings, such as web.config or .env files. It's quite normal for developers to check-in an .env.sample file instructing other developers who clone the repository to follow the structure of .env.sample when they create their own .env files.

This .env file is instructed to be ignored from the get go, but the .env.sample file is not. This means that I can continue to maintain .env.sample and keep my instructions updated, while the .env remains safely out of source control.

But this behavior can also be problematic sometimes. Let's say that you forgot to include an auto-generated folder such as node_modules in the first check-in. How do you now instruct Git to not track this folder going forward, even though you checked it in once?

First let's reset my repo to what's checked into remote.

```
git reset --hard && git pull
```

This command discards all of my changes and refreshes my local working copy from the remote location, just to make sure I have my teammates' changes on my disk.

Now I wish to instruct Git to stop tracking "afolder" but leave my working copy of afolder alone. This entire sequence can be seen in **Figure 19**.

There's a lot going on in **Figure 19**, so let's break it down step by step.

First, using the tree command, I show the current structure of my working tree. Note that my gitignore has instructed Git to ignore afolder and env.txt, but afolder\secondfile.md is already being tracked. The afolder\anewfile.md was created after the gitignore file was created, so it's not being tracked.

Next, I ask Git to remove files from the index recursively using the "-r" option, but using the --cached option, I instruct Git to remove files only from the index but leave the working tree alone. Long story short, this means: Leave my local files alone but fix the Git repo.

Running this command informs me of the changes Git made. It didn't remove the files from my disk though. To fully understand the changes, I then run a Git status command, which tells me that it deleted .gitignore, but not really—it now shows .gitignore as untracked. This means that now when I do a "git add .", those untracked files will now be tracked. But you know what won't be tracked? The afolder/secondfile.md won't be tracked going forward.

This achieves my goal of telling Git, hey, really, stop tracking this entire folder, just like my .gitignore instructs you to do.

To put things simply, simply adding a .gitignore won't cause Git to stop tracking files that are already being tracked

```
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn  main  tree
.
├── afolder
│   ├── anewfile.md
│   └── secondfile.md
├── dontreadme.md
└── env.txt

1 directory, 4 files
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn  main  git rm --cached -r .
rm '.gitignore'
rm 'afolder/secondfile.md'
rm 'dontreadme.md'
sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn  main +  git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    .gitignore
        deleted:    afolder/secondfile.md
        deleted:    dontreadme.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        dontreadme.md

sahilmalik@BigMac ~/Documents/Developer/temp/gitlearn  main +
```

Figure 19: Remove files you wish not to track

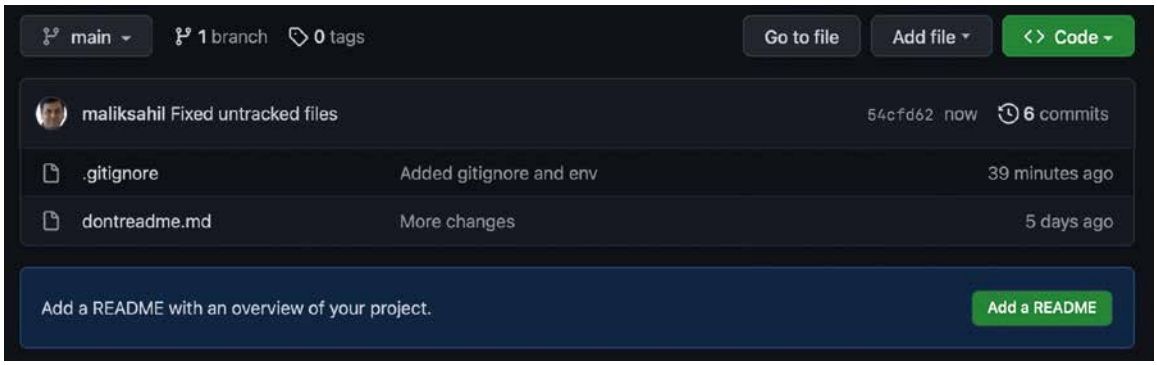


Figure 20: Git repo with my cleaned-up index

but match the .gitignore spec. This is by design. To actually untrack files, you also need to remove them from the index.

Now, go ahead and do a add, commit, and push. Now your Git repo should look like **Figure 20.**

You can imagine that "afolder" could be something like node_modules, or something that you actually wanted to get rid of.

Diff

When you're working on a software project, you're editing files. This is your source code, and you need plenty of things to help you keep control of what's being committed. You've already seen a Git command called Git status that lets you do this at file level. But what about changes inside a file? Perhaps you want a good way to compare two versions of a file and get a clear idea of what changes will be made if you push your changes.

Let's understand this with an example. The current state of my Git repo is shown in **Figure 21**.

As can be seen in **Figure 21**, I have one file in the root called "dontreadme.md", and a few other files and folders (mostly ignored by gitignore). What I wish to do is add some text to dontreadme.md and create a new file called readme.md.

```
echo 'even more stuff' >> dontreadme.md
echo 'brand new file' > readme.md
```

You can now run Git status to see what has changed. Here's a trick. The output of Git status can be quite wordy. If you want to see a quick shorthand output, which may be useful when you have a lot of files, use the following command:

```
git status -s
```

The output of this command looks like this:

```
M dontreadme.md
?? readme.md
```

This output tells you that the dontreadme.md file has been modified. But the readme.md file is unstaged.

Now, go ahead and stage dontreadme.md.

```
git add dontreadme.md
```

Run a Git status -s again. The output in text remains the same, but notice closely that the "M" by dontreadme.md has changed from red to green.

Now append some more text to dontreadme.md. Don't stage this newly appended content and run Git status -s again. This time you'll see that dontreadme.md status now says "MM", one M is green, and the other is red. This can be seen in **Figure 22.**

So now you have some content in remote, some content staged, and some in working copy, and all this content is slightly different from each other.

How do you get ahead of the differences between these three? The magic command is:

```
git diff
```

The output of this command can be seen in **Listing 1**. Let's be honest: This output is quite cryptic. Let's try to understand what this output means. This command compares your working copy to staged changes.

- The a/ and b/ are directories—not real directories, but a way to show you that a/ is index, and b/ is the working directory.

Listing 1: Output of git diff

```
diff --git a/dontreadme.md b/dontreadme.md
index b84a4f4..d41a9fb 100644
--- a/dontreadme.md
+++ b/dontreadme.md
@@ -1,3 +1,4 @@
 first file
 more changes
 even more stuff
+so much stuff
```

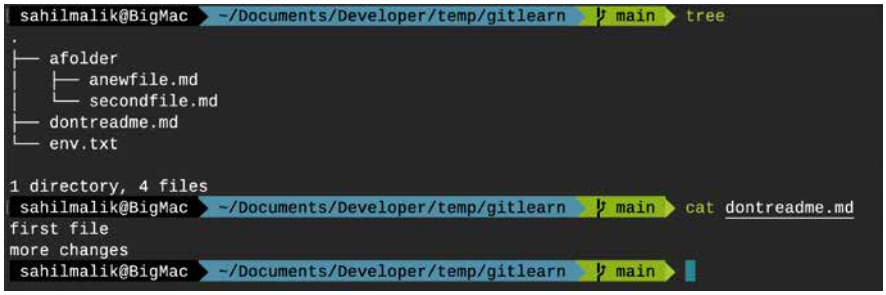


Figure 21: My Git repo's starting point

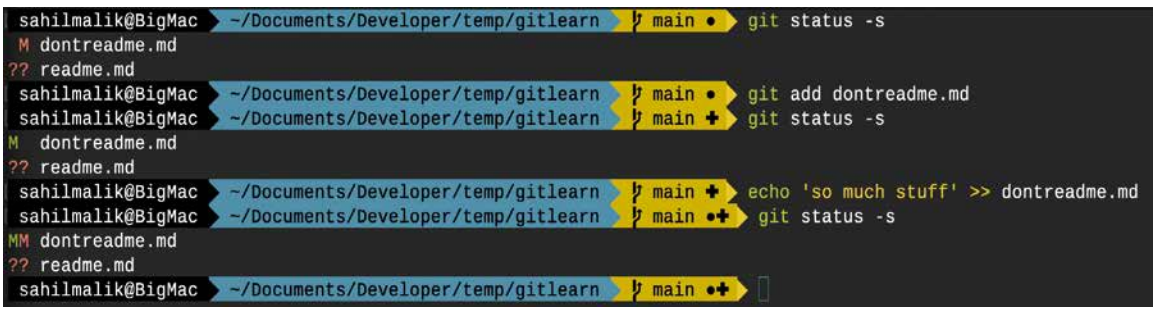


Figure 22: Git status -s.

- The IDs you see after that (b84a4f4) are BLOB IDs of the files mentioned.
- The 100644 you see is "mode bits", telling you that this isn't an executable file or a symlink; it's just a text file.
- The ---a/ +++b/ you see on the next line is interesting. The minus signs show lines in the a/ version but they are missing from the b/ version. And the plus signs show added lines in b/.
- The next line starting with @@ is also interesting. The changes are summarized as chunks, and here you have one chunk. This @@ line is the header of this chunk. It's telling you that starting at the first line, you have three files from the a/. And starting at line 1, you have four lines extracted.

It tries to color code it, but the color coding can frequently get messed up over ssh sessions or your local settings.

If you want to compare staged with remote, you simply use this command:

```
git diff --staged
```

Phew! This works, but it's tedious. Is there a better way?

Use VSCode as a Diffing Tool

In the real world, you use Git diffing tools. You can use any tool that supports diffing—cross-platform tools such as KDiff3, P4Merge, or, for Windows, you can use WinMerge. Personally, I prefer to use VSCode—it's a pretty nice diffing tool.

Most modern tools support this out of the box. You simply open a Git repo in VSCode and VSCode starts leveraging the output of Git behind the scenes to give you a visual Git experience. You can completely integrate diffing right inside of VSCode. Here is how.

First, instruct VSCode to act as the diffing tool for Git. To do so, edit your gitconfig file:

```
git config --global -e
```

Once your .gitconfig opens in your configured editor, add the following lines at the bottom of it:

```
[diff]
     tool = vscode
[difftool "vscode"]
     cmd = "code --wait --diff $LOCAL $REMOTE"
```

Now, instead of saying "git diff", run "git difftool". It should show you an output like this:

```
Viewing (1/1): 'dontreadme.md'
Launch 'vscode' [Y/n]?
```

If you hit "Y" on that prompt, it opens VSCode, which then takes care of showing you the diff. This can be seen in **Figure 23**.

You can also try "git difftool --staged" to view the staged diff.

This is a more visual diff. Using VSCode is so much easier to understand than viewing the ASCII wall that **git diff** threw at me.

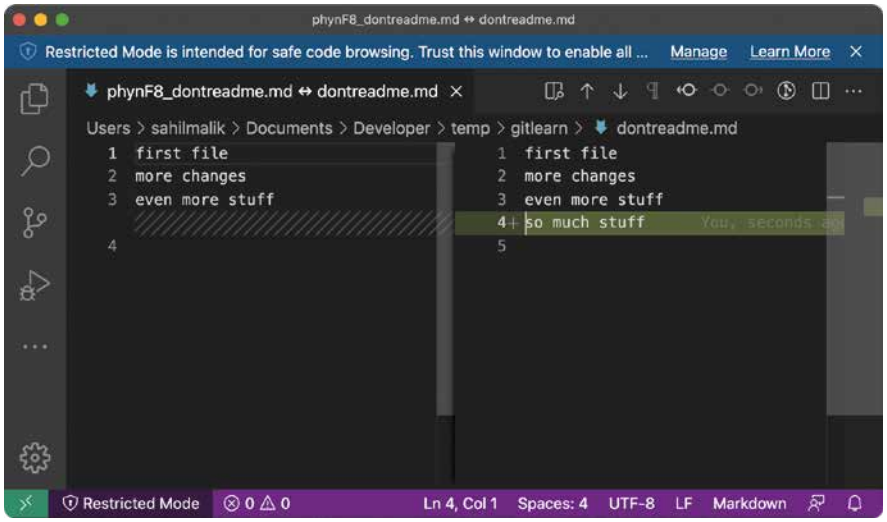


Figure 23: Here, I'm diffing in VSCode like a champ.

Summary

Git is an incredibly important skill. And let's be honest: There's a learning curve here. When I started writing this article, I thought I'd cover a bunch of interesting stuff that some developers consider advanced, such as merging, forking, concurrent developers working, and resolving merge conflicts. Those are skills that you'll need and use daily in a typical developer's workday. But as I started writing this article, I realized how much knowledge and how many nuances I take for granted, and before I knew it, this article started getting longer than I had anticipated.

Even to cover the basics of Git and tie it to practical real-world situations, there's an unsaid skill, assumed knowledge, that can be frustrating to discover.

I'm really curious to know: Do you consider yourself to be a seasoned developer? Do you use Git regularly? Even in these ultra-basic commands around Git usage, did you discover anything new? What complex Git situations would you like to see broken down in future articles? Do let me know.

git commit -m "That's a wrap" && git push.

Sahil Malik

Enhance Your MVC Applications Using JavaScript and jQuery: Part 3

This article continues my series on how to enhance the user experience (UX) of your MVC applications, and how to make them faster. In the first article, entitled Enhance Your MVC Applications Using JavaScript and jQuery: Part 1 (https://www.codemag.com/Article/2109031/Enhance-Your-MVC-Applications-Using-JavaScript-and-jQuery-Part-1), and the second article, entitled Enhance Your MVC Applications Using JavaScript and jQuery: Part 2 (https://www.codemag.com/Article/2111031/Enhance-Your-MVC-Applications-Using-JavaScript-and-jQuery-Part-2), you learned about the starting MVC application, which was coded using all server-side C#. You then added JavaScript and jQuery to avoid post-backs and enhance the UX in various ways. If you haven't already read these articles, I highly recommend that you read them to learn about the application you're enhancing in this series of articles.

Paul D. Sheriff
http://www.pdsa.com

Paul has been in the IT industry over 33 years. In that time, he has successfully assisted hundreds of companies to architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles, and speaking engagements at user groups and conferences around the world.
Paul has 23 courses in the www.pluralsight.com library (http://www.pluralsight.com/author/paul-sheriff) on topics ranging from JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap. Contact Paul at psheriff@pdsa.com.

In this article, you're going to build Web API calls that you can call from the application to avoid post-backs. You're going to add calls to add, update, and delete shopping cart information. In addition, you're going to learn to work with dependent drop-down lists to also avoid post-backs. Finally, you learn to use jQuery auto-complete instead of a drop-down list to provide more flexibility to your user.

The Problem: Adding to Shopping Cart Requires a Post-Back

On the Shopping page, each time you click on an Add to Cart button (**Figure 1**), a post-back occurs and the entire page is refreshed. This takes time and causes a flash on the page that can be annoying to the users of your site. In addition, it takes time to perform this post-back because all the data must be retrieved from the database server, the entire page needs to be rebuilt on the server side, and then the browser must redraw the entire page. All of this leads to a poor user experience.

The Solution: Create a Web API Call

The first thing to do is to create a new Web API controller to handle the calls for the shopping cart functionality. Right mouse-click on the PaulsAutoParts project and create a new folder named **ControllersApi**. Right mouse-click on the ControllersApi folder and add a new class named **ShoppingApiController.cs**. Remove the default code in the file and add the code shown in **Listing 1** to this new class file.

Add two attributes before this class definition to tell .NET that this is a Web API controller and not an MVC page controller. The [ApiController] attribute enables some features such as attribute routing, automatic model validation, and a few other API-specific behaviors. When using the [ApiController] attribute, you must also add the [Route] attribute. The route attribute adds the prefix "api" to the default "[controller]/[action]" route used by your MVC page controllers. You can choose whatever prefix you wish, but the "api" prefix is a standard convention that most developers use.

In the constructor for this API controller, inject the AppSession, and the product and vehicle type repositories. Assign the product and vehicle type repositories to the corresponding private read-only fields defined in this class.

The AddToCart() method is what's called from jQuery Ajax to insert a product into the shopping cart that's stored in the Session object. This code is similar to the code written in the MVC controller class ShoppingController.Add() method. After adding the **id** passed in by Ajax, a status code of 200 is passed back from this Web API call to indicate that the product was successfully added to the shopping cart. At this point, you have everything you need on the back-end to add a product to the shopping cart via an Ajax call.

Modify the Add to Cart Link

It's now time to modify the client-side code to take advantage of this new Web API method. You no longer want a post-back to occur when you click on the **Add to Cart** link, so you need to remove the **asp-** attributes and add code to make an Ajax call. Open the **Views\Shopping_ShoppingList.cshtml** file and locate the Add to Cart <a> tag and remove the **asp-action="Add"** and the **asp-route-id="@item.ProductId"** attributes. Add **id** and **data-** attributes, and an **onclick** event, as shown in the code snippet below.

Figure 1: Adding an item to the shopping cart can be more efficiently handled using Ajax.

Listing 1: Create a new Web API controller with methods to eliminate post-backs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using PaulsAutoParts.AppClasses;
using PaulsAutoParts.Common;
using PaulsAutoParts.EntityLayer;
using PaulsAutoParts.ViewModelLayer;

namespace PaulsAutoParts.ControllersApi
{
  [ApiController]
  [Route("api/[controller]/[action]")]
  public class ShoppingApiController : AppController
  {
    #region Constructor
    public ShoppingApiController(AppSession session,
            IRepository<Product, ProductSearch> repo,
            IRepository<VehicleType,
              VehicleTypeSearch> vrepo) : base(session)
    {
      _repo = repo;
      _vehicleRepo = vrepo;
    }
    #endregion

    #region Private Fields
    private readonly IRepository<Product, ProductSearch> _repo;
    private readonly IRepository<VehicleType,
                                 VehicleTypeSearch> _vehicleRepo;
    #endregion

    #region AddToCart Method
    [HttpPost(Name = "AddToCart")]
    public IActionResult AddToCart([FromBody]int id)
    {
      // Set Cart from Session
      ShoppingViewModel vm = new(_repo, _vehicleRepo,
        UserSession.Cart);

      // Set "Common" View Model Properties from Session
      base.SetViewModelFromSession(vm, UserSession);

      // Add item to cart
      vm.AddToCart(id, UserSession.CustomerId.Value);

      // Set cart into session
      UserSession.Cart = vm.Cart;

      return StatusCode(StatusCodes.Status200OK, true);
    }
    #endregion
  }
}
```

```
<a class="btn btn-info"
   id="updateCart"
   data-isadding="true"
   onclick="pageController
    .modifyCart(@item.ProductId, this)">
  Add to Cart
</a>
```

When you post back to the server, a variable in the view model class is set on each product to either display the Add to Cart link or the Remove from Cart link. When using client-side code, you're going to toggle the same link to either perform the add or the remove. Use the **data-isadding** attribute on the anchor tag to determine whether you're doing an add or a remove.

Add Code to Page Closure

The **onclick** event in the anchor tag calls a method on the pageController called modifyCart(). You pass to this cart the current product ID and a reference to the anchor tag itself. Add this modifyCart() method by opening the **Views\Shopping\Index.cshtml** file and adding the three private methods (**Listing 2**) to the **pageController** closure: modifyCart(), addToCart(), and removeFromCart(). The modifyCart() method is the one that's made public; the other two are called by the modifyCart() method.

The modifyCart() method checks the value in the **data-isadding** attribute to see if it's true or false. If it's true, call the addToCart() method, change the link text to "Remove from Cart", set the **data-isadding="false"**, remove the class "btn-info", and add the class "btn-danger". If false, call the removeFromCart() method and change the attributes on the link to the opposite of what you just set. Modify the **return** object to expose the modifyCart() method.

```
return {
  "setSearchArea": setSearchArea,
  "modifyCart": modifyCart
}
```

Listing 2: Add three methods in the pageController closure to modify the shopping cart

```
function modifyCart(id, ctl) {
  // Are we adding or removing?
  if (Boolean($(ctl).data("isadding"))) {
    // Add product to cart
    addToCart(id);
    // Change the button
    $(ctl).text("Remove from Cart");
    $(ctl).data("isadding", false);
    $(ctl).removeClass("btn-info")
     .addClass("btn-danger");
  }
  else {
    // Remove product from cart
    removeFromCart(id);
    // Change the button
    $(ctl).text("Add to Cart");
    $(ctl).data("isadding", true);
    $(ctl).removeClass("btn-danger")
     .addClass("btn-info");
  }
}

function addToCart(id) {
}

function removeFromCart(id) {
}
```

Create the addToCart() Method

Write the addToCart() method in the **pageController** closure to call the new AddToCart() method you added in the ShoppingApiController class. Because you're performing a post, you may use either the jQuery $.ajax() or $.post() methods. I chose to use the $.post() method in the code shown in following snippet.

```
function addToCart(id) {
  let settings = {
    url: "/api/ShoppingApi/AddToCart",
    contentType: "application/json",
    data: JSON.stringify(id)
  }
  $.post(settings)
```

```
    .done(function (data) {
      console.log(
        "Product Added to Shopping Cart");
    })
    .fail(function (error) {
      console.error(error);
    });
}
```

Try It Out

Run the application and click on the Shop menu. Perform a search to display products on the Shopping page. Click on one of the **Add to Cart** links to add a product to the shopping cart. You should notice that the link changes to **Remove from Cart** immediately. Click on the "0 Items in Cart" link in the menu bar and you should see an item in the cart. Don't worry about the "0 Items in Cart" link; you'll fix that a little later in this article.

Figure 2: Removing items from a cart can be more efficiently handled using Ajax.

The Problem: Delete from Shopping Cart Requires a Post-Back

Now that you've added a product to the shopping cart using Ajax, it would be good to also remove an item from the cart using Ajax. The link on the product you just added to the cart should now be displaying **Remove from Cart** (**Figure 2**). This was set via the JavaScript you wrote in the addToCart() method. The **data-isadding** attribute has been set to a false value, so when you click on the link again, the code in the modifyCart() method calls the removeFromCart() method.

The Solution: Write Web API Method to Delete from the Shopping Cart

Open the **ControllersApi\ShoppingApiController.cs** file and add a new method named RemoveFromCart(), as shown in **Listing 3**. This method is similar to the Remove() method contained in the ShoppingController MVC class. A product ID is passed into this method and the RemoveFromCart() method is called on the view model to remove this product from the shopping cart help in the Session object. A status code of 200 is returned from this method to indicate that the product was successfully removed from the shopping cart.

Modify the Remove from Cart Link

You no longer want a post-back to occur when you click on the Remove from Cart link, so you need to remove the **asp-** attributes and add code to make an Ajax call. Open the **Views\Shopping_ShoppingList.cshtml** file and locate the Remove from Cart <a> tag and remove the *asp-action="Remove"* and the *asp-route-id="@item.ProductId"* attributes. Add an **id** and **data-** attributes, and an **onclick** event, as shown in the code below.

```
<a class="btn btn-danger"
   id="updateCart"
   data-isadding="false"
   onclick="pageController
    .modifyCart(@item.ProductId, this)">
  Remove from Cart
</a>
```

Notice that you're setting the **id** attribute to the same value as on the Add to Cart button. As you know, you can't have two HTML elements with the **id** attribute set to the same value, because these two buttons are wrapped within an @if() statement, and only one is written by the server into the DOM at a time.

Add Code to pageController

Open the **Views\Shopping\Index.cshtml** file and add code to the removeFromCart() method. Call the $.ajax() method by setting the **url** property to the location of the RemoveFromCart() method you added, and set the **type** property to "DELETE". Pass the **id** of the product to delete on the URL line, as shown in the following code snippet.

```
function removeFromCart(id) {
  $.ajax({
    url: "/api/ShoppingApi/RemoveFromCart/" + id,
    type: "DELETE"
  })
```

Listing 3: The RemoveFromCart() method deletes a product from the shopping cart

```
[HttpDelete("{id}", Name = "RemoveFromCart")]
public IActionResult RemoveFromCart(int id)
{
  // Set Cart from Session
  ShoppingViewModel vm = new(_repo,
    _vehicleRepo, UserSession.Cart);

  // Set "Common" View Model Properties
  // from Session
  base.SetViewModelFromSession(vm,
    UserSession);

  // Remove item to cart
  vm.RemoveFromCart(vm.Cart, id,
    UserSession.CustomerId.Value);

  // Set cart into session
  UserSession.Cart = vm.Cart;

  return StatusCode(StatusCodes.Status200OK,
    true);
}
```

```
    .done(function (data) {
      console.log(
        "Product Removed from Shopping Cart");
    })
    .fail(function (error) {
      console.error(error);
    });
}
```

Try It Out

Run the application and click on the Shop menu. Perform a search to display products on the Shopping page. Click on one of the **Add to Cart** links to add a product to the shopping cart. You should notice that the link changes to **Remove from Cart** immediately. Click on the "0 Items in Cart" link in the menu bar and you should see an item in the cart. Click on the back button on your browser and click the **Remove from Cart** link on the item you just added. Click on the "0 Items in Cart" link and you should see that there are no longer any items in the shopping cart.

The Problem: The "n Items in Cart" Link isn't Updated

After modifying the code in the previous section to add and remove items from the shopping cart using Ajax, you noticed that the "0 Items in Cart" link in the menu bar isn't updating with the current number of items in the cart. That's because this link is generated by data from the server-side. Because you're bypassing server-side processing with Ajax calls, you need to update this link yourself.

The Solution: Add Client-Side Code to Update Link

Open the **Views\Shared_Layout.cshtml** file and locate the "Items in Cart" link. Add an **id** attribute to the <a> tag and assign it the value of "itemsInCart", as shown in the following code snippet.

```
<a id="itemsInCart"
   class="text-light"
   asp-action="Index"
   asp-controller="Cart">
   @ViewData["ItemsInCart"] Items in Cart
</a>
```

Create a new method to increment or decrement the "Items in Cart" link. Open the **wwwroot\js\site.js** file and add a new method named modifyItemsInCartText() to the **mainController** closure, as shown in **Listing 4**. An argument is passed to this method to specify whether you're adding or removing an item from the shopping cart. This tells the method to either increment the number or decrement the number of items in the text displayed on the menu.

The modifyItemsInCartText() method extracts the text portion from the <a> tag holding the "0 Items in Cart". It calculates the position of the first space in the text, which allows you to parse the numeric portion, turn that into an integer, and place it into the variable named **count**. If the value passed into the **isAdding** parameter is true, then **count** is incremented by one. If the value passed is false, then **count** is decremented by one. The new numeric value is then placed where the old numeric value was in the string and this new string is inserted back into the <a> tag. Expose the modifyItemsInCartText()method from the **return** object on the **mainController** closure, as shown in the following code.

Listing 4: Add a modifyItemsInCartText() method to the mainController closure

```
function modifyItemsInCartText(isAdding) {
  // Get text from <a> tag
  let value = $("#itemsInCart").text();
  let count = 0;
  let pos = 0;

  // Find the space in the text
  pos = value.indexOf(" ");
  // Get the total # of items
  count = parseInt(value.substring(0, pos));

  // Increment or Decrement the total # of items
  if (isAdding) {
    count++;
  }
  else {
    count--;
  }

  // Create the text with the new count
  value = count.toString() + " " + value.substring(pos);
  // Put text back into the cart
  $("#itemsInCart").text(value);
}
```

```
return {
  "pleaseWait": pleaseWait,
  "disableAllClicks": disableAllClicks,
  "setSearchValues": setSearchValues,
  "isSearchFilledIn": isSearchFilledIn,
  "setSearchArea": setSearchArea,
  "modifyItemsInCartText": modifyItemsInCartText
}
```

Call this function after making the Ajax call to either add or remove an item from the cart. Open the **Views\Shopping\Index.cshtml** file and locate the done() method in the addToCart() method. Add the line shown just before the console.log() statement.

```
$.post(settings)
  .done(function (data) {
    mainController.modifyItemsInCartText(true);
    console.log("Product Added to Shopping Cart");
  })
// REST OF THE CODE HERE
```

Locate the done() method in the removeFromCart() method and add the line of code just before the console.log() statement, as shown in the following code snippet.

```
$.ajax({
  url: "/api/ShoppingApi/RemoveFromCart/" + id,
  type: "DELETE"
})
  .done(function (data) {
    mainController.modifyItemsInCartText(false);
    console.log("Product Removed from Shopping Cart");
  })
// REST OF THE CODE HERE
```

Try It Out

Run the application and click on the Shop menu. Perform a search to display products on the Shopping page. Click on one of the **Add to Cart** links to add a product to the shopping cart and notice the link changes to **Remove from Cart** immediately. You should also see the "Items in Cart" link increment. Click on the **Remove from Cart** link and you should see the "Items in Cart" link decrement.

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Articles" from the Category drop-down. Then select "Enhance your MVC Applications using JavaScript and jQuery: Part 3" from the Item drop-down.

The Problem: Dependent Drop-Downs Requires Multiple Post-Backs

A common user interface problem to solve is that when you choose an item from a drop-down, you then need a drop-down immediately following to be filled with information specific to that selected item. For example, run the application and select the Shop menu to get to the Shopping page. In the left-hand search area, select a Vehicle Year from the drop-down list (**Figure 3**). Notice that a post-back occurs and now a list of Vehicle Makes are filled into the corresponding drop-down. Once you choose a make, another post-back occurs and a list of vehicle models is filled into the last drop-down. Notice the flashing of the page that occurs each time you change the year or make caused by the post-back.

The Solution: Connect All Drop-Downs to Web API Services

To eliminate this flashing, create Web API calls to return makes and models. After selecting a year from the Vehicle Year drop-down, an Ajax call is made to retrieve all makes for that year in a JSON format. Use jQuery to build a new set of <option> objects for the Vehicle Make drop-down. The same process can be done for the Vehicle Model drop-down as well.

Open the **ControllersApi\ShoppingApiController.cs** file and add a new method named GetMakes() to get all makes of vehicles for a specific year, as shown in **Listing 5**. This method accepts the year of the vehicle to search for. The GetMakes() method on the ShoppingViewModel class is called to set the **Makes** property with the collection of vehicle makes that are valid for that year. The set of vehicle makes is returned from this Web API method.

Listing 5: The GetMakes() method returns all vehicles makes for a specific year

```
[HttpGet("{year}", Name = "GetMakes")]
public IActionResult GetMakes(int year)
{
  IActionResult ret;

  // Create view model
  ShoppingViewModel vm = new(_repo,
    _vehicleRepo, UserSession.Cart);

  // Get vehicle makes for the year
  vm.GetMakes(year);

  // Return all Makes
  ret = StatusCode(StatusCodes.Status200OK,
    vm.Makes);

  return ret;
}
```

Listing 6: The GetModels() method returns all vehicle models for a specific year and model

```
[HttpGet("{year}/{make}", Name = "GetModels")]
public IActionResult GetModels(int year,
  string make)
{
  IActionResult ret;

  // Create view model
  ShoppingViewModel vm = new(_repo,
    _vehicleRepo, UserSession.Cart);

  // Get vehicle models for the year/make
  vm.GetModels(year, make);

  // Return all Models
  ret = StatusCode(StatusCodes.Status200OK,
    vm.Models);

  return ret;
}
```

Next, add another new method named GetModels() to the ShoppingApiController class to retrieve all models for a specific year and make as shown in **Listing 6**. In this method, both a year and a vehicle make are passed in. The GetModels() method on the ShoppingViewModel class is called to populate the **Models** property with all vehicle models for that specific year and make. The collection of vehicle models is returned from this Web API method.

Modify Shopping Cart Page

It's now time to add a couple of methods to your shopping cart page to call these new Web API methods you added to the ShoppingApiController class. Open the **Views\Shopping\Index.cshtml** file and add a method to the **pageController** named getMakes(), as shown in **Listing 7**.

The getMakes() method retrieves the year selected by the user. It then clears the drop-down that holds all vehicle makes and the one that holds all vehicle models. Next, a call is made to the GetMakes() Web API method using the $.get() shorthand method. If the call is successful, use the jQuery each() method on the data returned to iterate over the collection of vehicle makes returned. For each make, build an <option> element with the vehicle make within the <option> and append that to the drop-down.

Add another method to the **pageController** named getModels(), as shown in **Listing 8**. The getModels() method retrieves both the year and make selected by the user. Clear the models drop-down list in preparation for loading the new list. Call the GetModels() method using the $.get() shorthand method. If the call is successful, use the jQuery

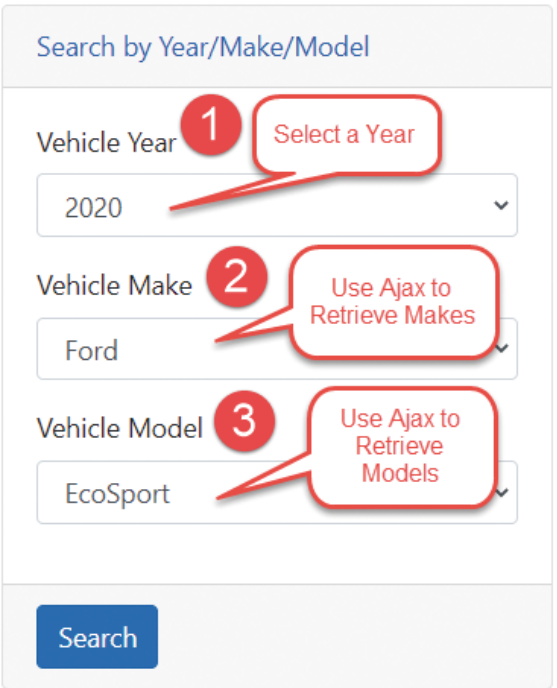


Figure 3: Multiple post-backs occur when you select a different value from any of these drop-downs.

each() method on the data returned to iterate over the collection of vehicle models returned. For each model, build an <option> element with the vehicle model within the <option> and append that to the drop-down.

Because you added two new private methods to the **pageController** closure, you need to expose these two methods by modifying the **return** object, as shown in the following code snippet.

```
return {
  "setSearchArea": setSearchArea,
  "modifyCart": modifyCart,
  "getMakes": getMakes,
  "getModels": getModels
}
```

Now that you have the new methods written and exposed from your pageController closure, hook them up to the appropriate **onchange** events of the drop-downs for the year and make within the search area on the page. Locate the <select> element for the **SearchEntity.Year** property and modify the **onchange** event to look like the following code snippet.

```
<select class="form-control"
  onchange="pageController.getMakes(this);"
  asp-for="SearchEntity.Year"
  asp-items="@(new SelectList(Model.Years))">
</select>
```

Next, locate the <select> element for the *SearchEntity.Make* property and modify the **onchange** event to look like the following code snippet.

```
<select class="form-control"
  onchange="pageController.getModels(this);"
  asp-for="SearchEntity.Make"
  asp-items="@(new SelectList(Model.Makes))">
</select>
```

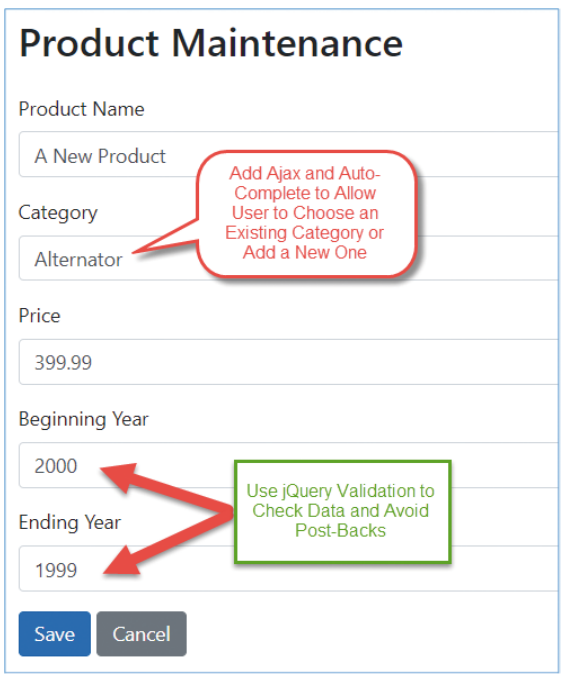


Figure 4: jQuery has validation capabilities as well as an auto-complete that can make your UI more responsive and avoid post-backs

Try It Out

Run the application and click on the Shop menu. Expand the "Search by Year/Make/Model" search area and select a year from the drop-down. The vehicle makes are now filled into the drop-down, but the page didn't flash because there's no longer a post-back. If you select a vehicle make from the drop-down, you should see the vehicle models filled in, but again, the page didn't flash because there was no post-back.

The Problem: Allow a User to Either Select an Existing Category or Add a New One

Click on the **Admin > Products** menu, then click the Add button to allow you to enter a new product (**Figure 4**). Notice that the Category field is a text box. This is fine if you want to add a new Category, but what if you want the user

Listing 7: The getMakes() method retrieves vehicle makes and builds a drop-down

```
function getMakes(ctl) {
  // Get year selected
  let year = $(ctl).val();

  // Search for element just one time
  let elem = $("#SearchEntity_Make");

  // Clear makes drop-down
  elem.empty();
  // Clear models drop-down
  $("#SearchEntity_Model").empty();

  $.get("/api/ShoppingApi/GetMakes/" +
    year, function (data) {
    // Load the makes into drop-down
    $(data).each(function () {
      elem.append(`<option>${this}</option>`);
    });
  })
    .fail(function (error) {
      console.error(error);
    });
}
```

Listing 8: The getModels() method retrieves vehicle models and builds a drop-down

```
function getModels(ctl) {
  // Get currently selected year
  let year = $("#SearchEntity_Year").val();
  // Get model selected
  let model = $(ctl).val();

  // Search for element just one time
  let elem = $("#SearchEntity_Model");

  // Clear models drop-down
  elem.empty();

  $.get("/api/ShoppingApi/GetModels/" +
    year + "/" + model, function (data) {
    // Load the makes into drop-down
    $(data).each(function () {
      elem.append(`<option>${this}</option>`);
    });
  })
    .fail(function (error) {
      console.error(error);
    });
}
```

to be able to select from the existing categories already assigned to products? You could switch this to a drop-down list, but then the user could only select an existing category and wouldn't be able to add a new one on the fly. What would be ideal is to use a text box, but also have a drop-down component that shows them the existing categories as they type in a few letters into the text box.

The Solution: Use jQuery UI Auto-Complete

To solve this problem, you need to bring in the jQuery UI library and use the auto-complete functionality. Once added to your project, connect a jQuery auto-complete to the Category text box so after the user starts to type, a list of existing categories can be displayed directly under the text box.

Modify the Product Repository Class

First, you need to make some changes to the back-end to support searching for categories by finding where a category starts with the text the user types in. Open the **RepositoryClasses\ProductRepository.cs** file in the **PaulsAutoParts.DataLayer** project and add a new method named SearchCategories() to this class. This method takes the characters entered by the user and queries the database to retrieve only those categories that start with those characters, as shown in the following code snippet.

```
public List<string> SearchCategories(
  string searchValue)
{
  return _DbContext.Products
    .Select(p => p.Category).Distinct()
    .Where(p => p.StartsWith(searchValue))
    .OrderBy(p => p).ToList();
}
```

This LINQ query roughly translates to the following SQL query.

```
SELECT DISTINCT Category
FROM Product
WHERE Category LIKE 'C%'
```

Listing 9: The SearchCategories() method performs a search for categories based on user input

```
[HttpGet("{searchValue}",
  Name = "SearchCategories")]
public IActionResult SearchCategories(
  string searchValue)
{
  IActionResult ret;

  ShoppingViewModel vm = new(_repo,
    _vehicleRepo, UserSession.Cart);

  if (string.IsNullOrEmpty(searchValue)) {
    // Get all product categories
    vm.GetCategories();

    // Return all categories
    ret = StatusCode(StatusCodes.Status200OK,
      vm.Categories);
  }
  else {
    // Search for categories
    ret = StatusCode(StatusCodes.Status200OK,
      vm.SearchCategories(searchValue));
  }

  return ret;
}
```

Modify the Shopping View Model Class

Instead of calling the repository methods directly from controller classes, it's best to let your view model class call these methods. Open the **ShoppingViewModel.cs** file in the **PaulsAutoParts.ViewModeLayer** project. Add a new method to this class named SearchCategories() that makes the call to the repository class method you just created.

```
public List<string> SearchCategories(
  string searchValue)
{
  return ((ProductRepository)Repository)
    .SearchCategories(searchValue);
}
```

Add New Web API to Controller

It's now time to create the Web API method for you to call via Ajax to search for the categories based on each character the user types into the text box. Open the **ControllersApi\ShoppingApiController.cs** file and add a new method named SearchCategories() to this controller class, as shown in **Listing 9**. This method accepts the character(s) typed into the Category text box; if it's blank, it returns all categories, and otherwise passes the search value to the SearchCategories() method you just created.

Add jQuery UI Code to Product Page

Open the **Views\Product\ProductIndex.cshtml** file and at the top of the page, just below the setting of the page title, add a new section to include the **jquery-ui.css** file. This is needed for styling the auto-complete drop-down. Please note that for the limits of printing this article, I had to break the **href** attribute on to two lines. When you put this into your cshtml file, be sure to put it all on one line.

```
@section HeadStyles {
  <link rel="stylesheet"
    href="//code.jquery.com/ui/1.12.1
          /themes/base/jquery-ui.css">
}
```

Now go to the bottom of the file and in the **@section Scripts** and just before your opening <script> tag, add the following <script> tag to include the jQuery UI JavaScript file. Please note that for printing this article, I had to break the **src** attribute into two lines. When you put this into your cshtml file, be sure to put it all on one line.

```
<script src="https://code.jquery.com/ui
            /1.12.1/jquery-ui.js">
</script>
```

Add a new method to the **pageController** closure named categoryAutoComplete(). The categoryAutoComplete() method is the publicly exposed method that's called from the $(document).ready() to hook up the auto-complete to the category text box using the autocomplete() method. Pass in an object to the autocomplete() method to set the **source** property to a method named searchCategories(), which is called to retrieve the category data to display in the drop-down under the text box. The **minLength** property is set to the minimum number of characters that must be typed prior to making the first call to the searchCategories() method. I've set it to one, so the user must type in at least one character in order to have the searchCategories() method called.

```
function categoryAutoComplete() {
  // Hook up Category auto-complete
  $("#SelectedEntity_Category").autocomplete({
    source: searchCategories,
    minLength: 1
  });
}
```

Add the searchCategories() method that's called from the **source** property. This method must accept a **request** object and a **response** callback function. This method uses the $.get() method to make the Web API call to the SearchCategories() method passing in the **request.term** property, which is the text the user entered into the category text box. If the call is successful, the **data** retrieved back from the Ajax call is sent back via the **response** callback function.

```
function searchCategories(request, response) {
  $.get("/api/ShoppingApi/SearchCategories/" +
    request.term, function (data) {
    response(data);
  })
    .fail(function (error) {
      console.error(error);
    });
}
```

Modify the return object to expose the categoryAutoComplete() method.

```
return {
  "setSearchValues": setSearchValues,
  "setSearchArea": mainController.setSearchArea,
  "isSearchFilledIn":
     mainController.isSearchFilledIn,
  "categoryAutoComplete": categoryAutoComplete
}
```

Finally, modify the $(document).ready() function to call the **pageController.categoryAutoComplete()** method to hook up jQuery UI to the Category text box.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();

  // Hook up category auto-complete
  pageController.categoryAutoComplete();

  // Collapse search area or not?
  pageController.setSearchValues();
  // Initialize search area on this page
  pageController.setSearchArea();
});
```

Try It Out

Run the application and select the **Admin > Products** menu. Click on the **Add** button and click into the Category text box. Type the letter **T** in the Category input field and you should see a drop-down appear of categories that start with the letter T.

Vehicle Type Page Needs Auto-Complete for Vehicle Make Text Box

There's another page in the Web application that can benefit from the jQuery UI auto-complete functionality. On the Vehicle Type maintenance page, when the user wants to add a new vehicle, they should be able to either add a new make or select from an existing one.

Add New Method to Vehicle Type Repository

Let's start with modifying the code on the server to support searching for vehicle makes. Open the **RepositoryClasses\VehicleTypeRepository.cs** file and add a new method named SearchMakes(), as shown in the following code snippet.

```
public List<string> SearchMakes(string make)
{
  return _DbContext.VehicleTypes
         .Select(v => v.Make).Distinct()
         .Where(v => v.StartsWith(make))
         .OrderBy(v => v).ToList();
}
```

This LINQ query roughly translates to the following SQL query:

```
SELECT DISTINCT Make
FROM Lookup.VehicleType
WHERE Make LIKE 'C%'
```

Add a New Method to Vehicle Type View Model Class

Instead of calling the repository methods directly from controller classes, it's best to let your view model class call

ADVERTISERS INDEX

Advertisers Index

CODE Consulting	www.codemag.com/code	7
CODE Consulting	www.codemag.com/onehourconsulting	75
CODE Legacy	www.codemag.com/modernize	76
DevIntersection	www.devintersection.com	2
dtSearch	www.dtSearch.com	11
LEAD Technologies	www.leadtools.com	5
Live on Maui	www.live-on-maui.com	38

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.

Listing 10: Create a new Web API controller for handling vehicle type maintenance

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using PaulsAutoParts.AppClasses;
using PaulsAutoParts.Common;
using PaulsAutoParts.DataLayer;
using PaulsAutoParts.EntityLayer;
using PaulsAutoParts.ViewModelLayer;

namespace PaulsAutoParts.ControllersApi
{
  [ApiController]
  [Route("api/[controller]/[action]")]
  public class VehicleTypeApiController :
    AppController
  {
    #region Constructor
    public VehicleTypeApiController(
      AppSession session,
      IRepository<VehicleType,
      VehicleTypeSearch> repo) : base(session)
    {
      _repo = repo;
    }
    #endregion

    #region Private Fields
    private readonly IRepository<VehicleType,
      VehicleTypeSearch> _repo;
    #endregion

    #region SearchMakes Method
    [HttpGet("{make}", Name = "SearchMakes")]
    public IActionResult SearchMakes(string make)
    {
      IActionResult ret;

      VehicleTypeViewModel vm = new(_repo);

      // Return all makes found
      ret = StatusCode(StatusCodes.Status200OK,
        vm.SearchMakes(make));

      return ret;
    }
    #endregion
  }
}
```

these methods. Open the **VehicleTypeViewModel.cs** file in the **PaulsAutoParts.ViewModeLayer** project. Add a new method to this class named SearchMakes() that makes the call to the repository class method you just created.

```
public List<string> SearchMakes(
  string searchValue)
{
  return ((VehicleTypeRepository)Repository)
    .SearchMakes(searchValue);
}
```

Create New API Controller Class

Add a new class under the ControllersApi folder named **VehicleTypeApiController.cs**. Replace all the code within the new file with the code shown in **Listing 10**. Most of this code is boiler-plate for a Web API controller. The important piece is the SearchMakes() method that's going to be called from jQuery Ajax to perform the auto-complete.

Add jQuery UI Code to Vehicle Type Page

Open the **Views\VehicleType\VehicleTypeIndex.cshtml** file and, at the top of the page, just below the setting of the page title, add a new section to include the **jquery-ui.css** file. This is needed for styling the auto-complete drop-down.

```
@section HeadStyles {
  <link rel="stylesheet"
    href="//code.jquery.com/ui/1.12.1
          /themes/base/jquery-ui.css">
}
```

Now go to the bottom of the file and in the **@section Scripts** and just before your opening <script> tag, add the following <script> tag to include the jQuery UI JavaScript file.

```
<script src="https://code.jquery.com/ui
        /1.12.1/jquery-ui.js">
</script>
```

Add a new method to the **pageController** closure named makesAutoComplete(). The makesAutoComplete() method is the publicly exposed method that is called from the $(document).ready() to hook up the jQuery auto-complete to the vehicle makes text box, just like you did for hooking up the category text box. Pass in an object to this method that sets the **source** property to a method named searchMake(), which is called to retrieve the vehicle makes to display in the drop-down under the text box. The **minLength** property is set to the minimum number of characters that must be typed prior to making the first call to the searchMakes() method. I've set it to one, so the user must type in at least one character in order to have the searchMakes() method called.

```
function makesAutoComplete() {
  // Hook up Makes auto-complete
  $("#SelectedEntity_Make").autocomplete({
    source: searchMakes,
    minLength: 1
  });
}
```

Add the searchMakes() method that is called from the **source** property. This method must accept a **request** object and a **response** callback function. This method uses the $.get() method to make the Web API call to the SearchMakes() method passing in the **request.term** property, which is the text the user entered into the vehicle makes text box. If the call is successful, the data retrieved back from the Ajax call is sent back via the **response** callback function.

```
function searchMakes(request, response) {
  $.get("/api/VehicleTypeApi/SearchMakes/" +
    request.term, function (data) {
    response(data);
  })
    .fail(function (error) {
      console.error(error);
    });
}
```

Modify the **return** object to expose the makesAutoComplete() method.

```
return {
  "setSearchValues": setSearchValues,
  "setSearchArea":
    mainController.setSearchArea,
  "isSearchFilledIn":
    mainController.isSearchFilledIn,
  "addValidationRules": addValidationRules,
  "makesAutoComplete": makesAutoComplete
}
```

Finally, modify the $(document).ready() function to call the **pageController.makesAutoComplete()** method to hook up jQuery UI to the vehicle makes text box.

```
$(document).ready(function () {
  // Add jQuery validation rules
  pageController.addValidationRules();

  // Hook up makes auto-complete
  pageController.makesAutoComplete();

  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  // Initialize search area on this page
  pageController.setSearchArea();
});
```

Try It Out

Run the application and select the **Admin > Vehicle Types** menu. Click on the Add button and click into the Makes text box. Type the letter **C** and you should see a drop-down appear of makes that start with the letter C.

Search by Multiple Fields in Auto-Complete

Technically, in the last sample, you should also pass the year that the user input to the vehicle makes auto-complete. However, just to keep things simple, I wanted to just pass in a single item. Let's now hook up the auto-complete for the vehicle model input. In this one, you're going to pass in the vehicle year, make, and the letter typed into the vehicle model text box to a Web API call from the auto-complete method.

Add a New Method to the Vehicle Type Repository

Open the **RepositoryClasses\VehicleTypeRepository.cs** file and add a new method named SearchModels(), as shown in the following code snippet. This method makes the call to the SQL Server to retrieve all distinct vehicle models for the specified year, make, and the first letter or two of the model passed in.

```
public List<string> SearchModels(
  int year, string make, string model)
{
  return _DbContext.VehicleTypes
    .Where(v => v.Year == year &&
      v.Make == make &&
      v.Model.StartsWith(model))
    .Select(v => v.Model).Distinct()
    .OrderBy(v => v).ToList();
}
```

Listing 11: The searchModels() method passes three values to the SearchModels() Web API

```
function searchModels(request, response) {
  let year = $("#SelectedEntity_Year").val();
  let make = $("#SelectedEntity_Make").val();

  if (make) {
    $.get("/api/VehicleTypeApi/
     SearchModels/" + year + "/" +
       make + "/" +
       request.term, function (data) {
      response(data);
    })
      .fail(function (error) {
        console.error(error);
      });
  }
  else {
    searchModels(request, response);
  }
}
```

Add aNew Method to the Vehicle Type View Model Class

Instead of calling repository methods directly from controller classes, it's best to let your view model class call these methods. Open the **VehicleTypeViewModel.cs** file in the **PaulsAutoParts.ViewModeLayer** project. Add a new method to this class named SearchModels() that makes the call to the repository class method you just created.

```
public List<string> SearchModels(int year,
   string make, string searchValue)
{
  return ((VehicleTypeRepository)Repository)
    .SearchModels(year, make, searchValue);
}
```

> Users will be grateful for AutoComplete.

Modify API Controller

Open the **ControllersApi\VehicleTypeApiController.cs** file and add a new method named SearchModels() that can be called from the client-side code. This method is passed the year, make, and model to search for. It initializes the VehicleTypeViewModel class and makes the call to the SearchModels() method to retrieve the list of models that match the criteria passed to this method.

```
[HttpGet("{year}/{make}/{model}",
  Name = "SearchModels")]
public IActionResult SearchModels(int year,
  string make, string model)
{
  IActionResult ret;
```

Listing 12: Hook up the auto-complete functionality by calling the appropriate method in the pageController closure

```
$(document).ready(function () {
  // Add jQuery validation rules
  pageController.addValidationRules();

  // Hook up makes auto-complete
  pageController.makesAutoComplete();

  // Hook up models auto-complete
  pageController.modelsAutoComplete();

  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  // Initialize search area on this page
  pageController.setSearchArea();
});
```


SPONSORED SIDEBAR:

Need FREE Project Advice? CODE Can Help!

No strings, free advice on new or existing software development projects. CODE Consulting experts have experience in cloud, Web, desktop, mobile, microservices, containers, and DevOps projects. Schedule your free hour of CODE call with our expert consultants today. For more information, visit www.codemag.com/consulting or email us at info@codemag.com.


```
  VehicleTypeViewModel vm = new(_repo);

  // Return all models found
  ret = StatusCode(StatusCodes.Status200OK,
    vm.SearchModels(year, make, model));

  return ret;
}
```

Modify the Page Controller Closure

Open the **Views\VehicleType\VehicleTypeIndex.cshtml** file. Add a new method to the **pageController** closure named modelsAutoComplete(). The modelsAutoComplete() method is the publicly exposed method called from the $(document).ready() to hook up the auto-complete to the models text box using the autocomplete() method. Pass in an object to this method to set the **source** property to the function to call to get the data to display in the drop-down under the text box. The **minLength** property is set to the minimum number of characters that must be typed prior to making the first call to the searchModels() function.

```
function modelsAutoComplete() {
  // Hook up Models AutoComplete
  $("#SelectedEntity_Model").autocomplete({
    source: searchModels,
    minLength: 1
  });
}
```

Add the searchModels() method (**Listing 11**) that's called from the **source** property. This method must accept a **request** object and a **response** callback function. This method uses the $.get() method to make the Web API call to the SearchModels() method passing in the year, make, and the **request.term** property, which is the text the user entered into the vehicle model text box. If the call is successful, the data retrieved back from the Ajax call is sent back via the **response** callback function.

Modify the **return** object to expose the modelsAutoComplete() method from the closure.

```
return {
  "setSearchValues": setSearchValues,
  "setSearchArea":
    mainController.setSearchArea,
  "isSearchFilledIn":
    mainController.isSearchFilledIn,
  "addValidationRules": addValidationRules,
  "makesAutoComplete": makesAutoComplete,
  "modelsAutoComplete": modelsAutoComplete
}
```

Modify the $(document).ready() function to make the call to the **pageController.modelsAutoComplete()** method, as shown in **Listing 12**.

Try It Out

Run the application and click on the **Admin > Vehicle Types** menu. Click on the Add button and put the value 2000 into the Year text box. Type/Select the value Chevrolet in the Makes text box. Type the letter **C** in the Models text box and you should see a few models appear.

> Calling Web API methods from jQuery Ajax can greatly speed up the performance of your Web pages

Summary

In this article, you once again added some functionality to improve the user experience of your website. Calling Web API methods from jQuery Ajax can greatly speed up the performance of your Web pages. Instead of having to perform a complete post-back and redraw the entire Web page, you can retrieve a small amount of data and update just a small portion of the Web page. Eliminating post-backs is probably one of the best ways to improve the user experience of your Web pages. Another technique you learned in this article was to take advantage of the jQuery UI auto-complete functionality.

Paul D. Sheriff

Software Development is Getting Harder, Not Easier

I hate that expression, "It's complicated." When people tell me it's complicated, it almost always isn't. What they're really saying is that they don't want to talk about it. But when we're talking about software development, it really IS complicated. Software development is notoriously difficult and it's getting steadily harder. And we need to talk about it. Imagine a world where a developer who understands the basics of programming (things like variables, branching, and looping) can spend a weekend reading a manual cover to cover and become productive in a new environment by Monday morning. An environment that includes built-in UI, database and reporting tools, its own IDE, integration paths for third-party systems, and everything else they might need to build and maintain a system. A world where the tool is only updated about once every two years or so and where reading a couple of articles or attending a conference for a few days gets them up to speed on ALL the new stuff. A world where developers get very skilled at using the tools and become masters of their craft.

That world existed between 20 and 30 years ago. Since then, the developer's world has accelerated and expanded at an increasing pace. Today, just keeping up with what's available to us is like drinking from a fire hose. Where do we even find time to write code?

> Before you go thinking this article is about nostalgia, it isn't.

I believe in the original definition of nostalgia, as a medical diagnosis for grave illness, one that couldn't be cured, even when the patient was lucky enough to be able to try to relive those memories. I believe we can't—and shouldn't—go back, only forward. But all hope is not lost. Let's start a conversation about complexity and begin to deal with it.

Where is complexity coming from? I believe it stems from the plethora of platforms and toolsets, higher expectations, larger systems, bigger teams, faster delivery of new ideas, and an ever-expanding set of opinions. Today, a full-stack developer must have a good working knowledge of multiple user interface tools, databases, cloud offerings, and business logic tools. Even picking just one UI platform, say HTML and JavaScript, there are hundreds of frameworks to choose from and they need to be mixed and matched to get a good result. New frameworks and major updates to existing frameworks are coming out almost daily. No one can make a perfect decision.

That's just for development; I didn't include new Web assembly-centric UIs like Blazor. I also ignored desktop and native mobile applications and more, and included choosing and learning an IDE to develop in. The number of platforms and toolsets available today is astounding and the pendulum is now swinging back for many companies who previously touted giving teams complete autonomy over their platforms and toolsets, to choosing and standardizing on a smaller subset. These companies are coping with too many choices, which leads to both decision paralysis and an inability to get everyone on the same page. Constraining the number of choices is one way to combat complexity.

> Too many choices leads to both decision paralysis and an inability to get everyone on the same page.

Higher expectations, larger systems, and larger teams are other drivers of complexity. Development is no longer just getting the code to function and doing some performance tuning. It also has to be secure, scalable, a great user experience, accessible from a variety of devices, cloud-deployable, tested, maintainable, kept up to date, and continuously improved. Also, at one time, most apps targeted internal users; now most apps are targeted at consumers, which ups the bar considerably.

The move toward both more systems and larger systems requires more teams and more developers, which leads to difficulties getting everyone rowing the boat in the same direction. When I say teams in this article, I don't just mean a development team within a company. I also mean the team developing the toolsets and platforms and those consuming what the team is building. We must work with teams both inside and outside our walls. This complexity can be mitigated by better collaboration.

The collaboration problem isn't unique to the software industry, and we're getting better at meeting these challenges, but the trends aren't going to slow down or reverse themselves.

The biggest drivers of complexity, in my experience, and those we can do the most about, are last two I mentioned. The faster delivery of new ideas and the ever-expanding set of opinions have, ironically, sprung from our own attempts to solve complexity issues and to make things easier. Today, everyone who solves an obstacle to coding productivity can publish and promote that solution. The solutions are often quite good and are easy to find on the Internet by anyone having a similar problem. However, sometimes the solution becomes widely accepted and is thought of as the "right way" to code.

Mike Yeager
www.internet.com

Mike is the CEO of EPS's Houston office and a skilled .NET developer. Mike excels at evaluating business requirements and turning them into results from development teams. He's been the Project Lead on many projects at EPS and promotes the use of modern best practices, such as the Agile development paradigm, use of design patterns, and test-drive and test-first development. Before coming to EPS, Mike was a business owner developing a high-profile software business in the leisure industry. He grew the business from two employees to over 30 before selling the company and looking for new challenges. Implementation experience includes .NET, SQL Server, Windows Azure, Microsoft Surface, and Visual FoxPro.

What our industry lacks is a good way to tell if a particular solution even applies to a project. We sometimes adopt things like SOLID principles, microservices, containers, Domain Driven Design, and container orchestration because they're considered "good" or they're thought of as a way to future-proof our solution, when in fact, they are often solutions to problems we don't have. In those cases, we're only adding complexity by adopting them.

How, then, can we be better at using the right tools at the right time? There's no magic solution, only hard work, careful reflection, and creativity. Let's examine the problem.

How to Solve the Problem of Complexity

When we think about how to solve the problem of complexity, the answer invariably comes down to breaking down a complex system into a series of smaller, more discrete, less complex systems that aren't too complex standing alone. This makes the new modules more approachable and the goals easier to achieve. But nothing comes for free and the catch is that we now have to communicate and coordinate among these smaller, more discreet systems, which often means new tools and new patterns, and that actually ADDS complexity to the system as a whole. It results in a new (but usually manageable) set of problems and more code to write, test, and maintain. Because of this, we have to be very careful in choosing how we approach each problem. It's time we stopped living by the dictum that every problem can be solved by adding another layer of abstraction and start asking if it **should** be solved that way.

What can we do about the complexity that comes from the steady gushing of new ideas? I believe that we, as an industry, should be more demanding of ourselves and of others. Instead of looking only at all of the good that comes from something new, we should demand to also know the downsides and demand help in determining when and where these ideas can be used to our advantage. Often, when looking at a new technology, the material is presented by diving in and showing how it works. There isn't even a mention of the tradeoffs or even which problems are being solved.

> Wouldn't it be refreshing to read about a new technology and why it was created, which things it does really well, and which things it isn't intended for instead of reading about how sliced bread has a new challenger?

New technologies should be approached with litmus tests and rules of thumb, even if we have to develop them ourselves. Is this new idea "good" or "bad"? It really depends on the problem you're trying to solve. What we need are better, more available ways to make that determination. Wouldn't it be refreshing to do some research into a new technology and read about why it was created, which things it does really well, and which things it isn't intended for, instead of reading about how sliced bread has a new challenger? I see this kind of documentation occasionally, but not nearly enough. It does us all a disservice to become such a fanboy of a technology that we don't present information that allows our peers to make good decisions. That information is at least as valuable as the sunshine we spread. Often more.

When thinking about the rapid growth of complexity and information in development, I often think about industries that have gone through this kind of evolution ahead of us. The airliner industry is a good example. The early airliner industry had a lot of competitors and the challenges were mainly engineering problems. Building commercial airplanes had been figured out and the challenge was in improving the airplanes. As the industry matured, the planes became increasingly larger and more complex and companies handled the complexity by adding more draftsmen, more engineers, more workstations, more bodies. They next handled the increased complexity by breaking the process down into smaller, more discreet processes. Some employees only worked on cabin interiors, some only worked on aerodynamics, some only on landing gear. All this segregation and specialization added complexities in other areas, like coordination among the teams and common goals. Hun-

leadr

People Want To Be Led & Developed, Not Managed.

Leadr is a people development software that helps you engage and grow every person on your team.

We're Hiring a Principal Graph Database Engineer

Remote, so long as you are living in the US

Apply Here

YOU WILL BE SET TO CRUSH THIS ROLE IF...

- You have 10+ years of experience building web applications, especially Sass applications.
- You have prior software engineering experience on a B2B or B2C SaaS product with a SDLC similar to what we are striving towards.
- You come from a startup background and are comfortable working in a fast-paced environment where you sometimes need to solve problems on your own, and you can drive a project to completion.
- You are a graph database expert, and are an expert in working with graph and other graph database solutions.
- You are highly proficient in other types of databases, such as relational and document database storage solutions.
- You have a masters degree or PhD in Information Systems.
- You can devise engineering solutions by considering multiple options and then present the pros/cons including cost, effort, complexity, scalability, maintainability, etc. as part of your proposal and walk others through your decision matrix.

Remuneration for this role is fixed at $120,000

Leadr is an Equal Opportunity-Affirmative Action Employer.

dreds of draftsmen with pencils were generating millions of drawings to be presented and argued over in endless meetings. New designs started to take years, sometimes decades, to come to fruition. Companies failed or were acquired by other companies.

The process had become too complex. This is where our industry is now. Developers are specializing more and we have lots of bodies all headed in different directions.

Companies like Boeing and Airbus innovated and began using software and collaboration tools that replaced much of the manual engineering, drafting, and meetings, and things began to speed up again, but not for the reason you may think. Software wasn't a magic bullet. All of that innovation also led to new capabilities, offsetting or even increasing complexity instead of reducing it. Todays, airliners are more complex than ever and the next generation will be even more complex.

What turned things around for the airliner manufacturers was that the teams and their software were getting better at working together; they created and adhered to standards and, most critically, they simplified the work other teams need to do to make use of what they'd built. The individual components were trending toward becoming black boxes that either did or didn't fit the requirements of a particular project and were relatively easy to make decisions about, whether that decision was to modify the black box or to build a new one. It wasn't the tooling or the innovation that made the defining difference. It was the ethos that each team was part of a larger team and that how those teams interacted was as important as what that team actually produced.

> Each team is part of a larger team and how those teams interact is as important as what that team produces.

As the software industry continues to evolve, the unfettered gushing of new ideas, tools, technologies, and platforms must become more stringent. Developers aren't keeping up. All of the wonderful toys have become as much a burden on the industry as a blessing. Software projects still fail at an astoundingly high rate and it's almost always because either the requirements are too complex or the chosen implementation is too complex.

> How teams interact is as important as what the team produces.

Complexity is killing us. Our industry needs to develop the airliner ethos that each team (in or out of your walls) is part of a larger team and that how those teams interact is as important as what the team produces. Our efforts must not be only to build newer, shinier, better, faster things, but to trend toward building black boxes that either do or don't fit the requirements of a particular project and are easy to reason about. We must allow others to make good, informed decisions about what we build.

This is where the next big innovation will happen. Evolving technologies, tools, platforms, ideas, and design patterns are good things, but they're only useful if they can be discovered, understood, and leveraged. I believe we need a self-curating system beyond the plethora of take-it-or-leave-it open-source projects and the commercially driven resources we have now. Our current situation has taken shape and that shape is a pile. And it's a very large and unruly pile that's overwhelming and doesn't serve our needs. We must organize that pile if we are to evolve.

Conclusion

There are things we can do to keep complexity from killing us. Currently, each of us is faced with a gigantic pile of new ideas, each labeled with a catchy name, some industry buzzwords and some marketing blurbs. The pile is too big and too messy. We need to change that so that each idea in the pile is labeled with its categorizations and specifications, so we can organize them and make rational and appropriate decisions about them. We need to specialize so that we can reduce the sheer number of ideas that we need to worry about and increase our understanding of and improve our discourse about those we do care about. We need to create better ways to collaborate with other specialists and their parts of the pile. We need to pick reasonable, curated defaults, so that not every specialist has to decide between every option in their part of the pile, every time. And finally, we need to develop a professional approach to these ideas so we can train the next generation, not just on looping and branching and technical basics, but also on how to make good use of the organized, categorized, curated, and well-described pile of ideas we're creating.

It is complicated. But there's hope.

Mike Yeager
CODE

Beginner's Guide to Deploying PHP Laravel on the Google Cloud Platform: Part 2

In the first article in this series (CODE Magazine, November/December 2021), you were introduced to Google Cloud Platform (GCP) and the PHP Laravel framework. You started by creating your first PHP Laravel project and pushed the app to a GitHub repository. Then you moved on to creating the Google App Engine (GAE) project and built the Google Cloud Build workflow to enable CI/CD (https://www.redhat.com/en/topics/devops/what-is-ci-cd) for the automated deployments on GCP.

Bilal Haidar
bhaidar@gmail.com
https://www.bhaidar.dev
@bhaidar

Bilal Haidar is an accomplished author, Microsoft MVP of 10 years, ASP.NET Insider, and has been writing for CODE Magazine since 2007.

With 15 years of extensive experience in Web development, Bilal is an expert in providing enterprise Web solutions.

He works at Consolidated Contractors Company in Athens, Greece as a full-stack senior developer.

Bilal offers technical consultancy for a variety of technologies including Nest JS, Angular, Vue JS, JavaScript and TypeScript.

Now, you'll go a step further and connect your app to a local MySQL database. Then, you'll introduce the Google Cloud SQL service and create your first SQL database in the cloud. Right after that, I'll show you one way to run Laravel database migrations from within the Cloud Build workflow. Finally, I'll enhance the Cloud Build workflow by showing you how to back up the SQL database every time you deploy a new version of the app on GCP.

First things first, let's locally connect the Laravel app to a MySQL database.

Create and Use a Local MySQL Database

In Part I of this series, I introduced Laravel Sail (https://laravel.com/docs/8.x/sail). It's a service offered by the Laravel team to dockerize your application locally. One of the containers that Sail creates locally, inside Docker, is the mysql container. It holds a running instance of MySQL Database Service. **Listing 1** shows the mysql service container section inside the docker-compose.yaml file.

When you start the Sail service, it creates a Docker container for the mysql service and automatically configures it with a MySQL Database that's ready to use in your application.

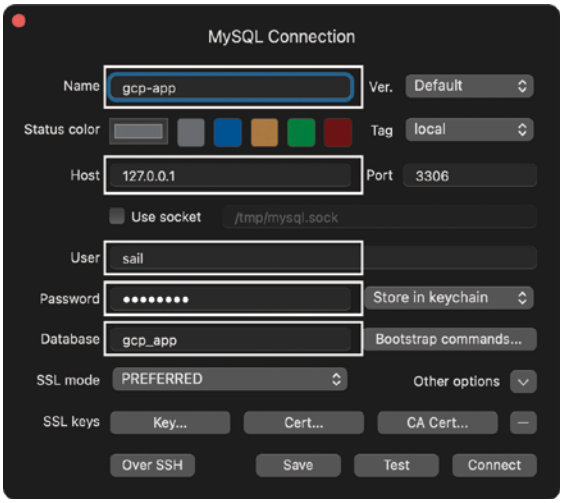


Figure 1: The TablePlus Connection Window

Sail picks up the database details from the current application environment variables that you usually define inside the .env file.

Let's have a look at the database section of the .env file:

```
DB_CONNECTION=mysql
B_HOST=mysql
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=sail
DB_PASSWORD=password
```

These are the default settings that ship with a new Laravel application using Sail service.

You can change the settings as you see fit. For now, I just changed the database name to be **gcp_app**.

> In Laravel 8.x, you can use any of the following database systems: MySQL 5.7+, PostgreSQL 9.6+, SQLite 3.8.8+, or SQL Server 2017.

Step 1: Create and Connect to a MySQL Database

For now, let's keep them as they are and start up the Docker containers using the Sail service command:

```
sail up --d
```

This command starts up all services that the docker-compose.yaml file hosts.

Let's connect to the database that sail has created. I'm using **TablePlus** (https://tableplus.com/) Database Management Tool to connect to the new database. Feel free to use any other management tool of your own preference. **Figure 1** shows the database connection window.

I've highlighted the important fields that need your attention:

- Name: The name of the connection
- Host: The IP address of the server hosting the MySQL database

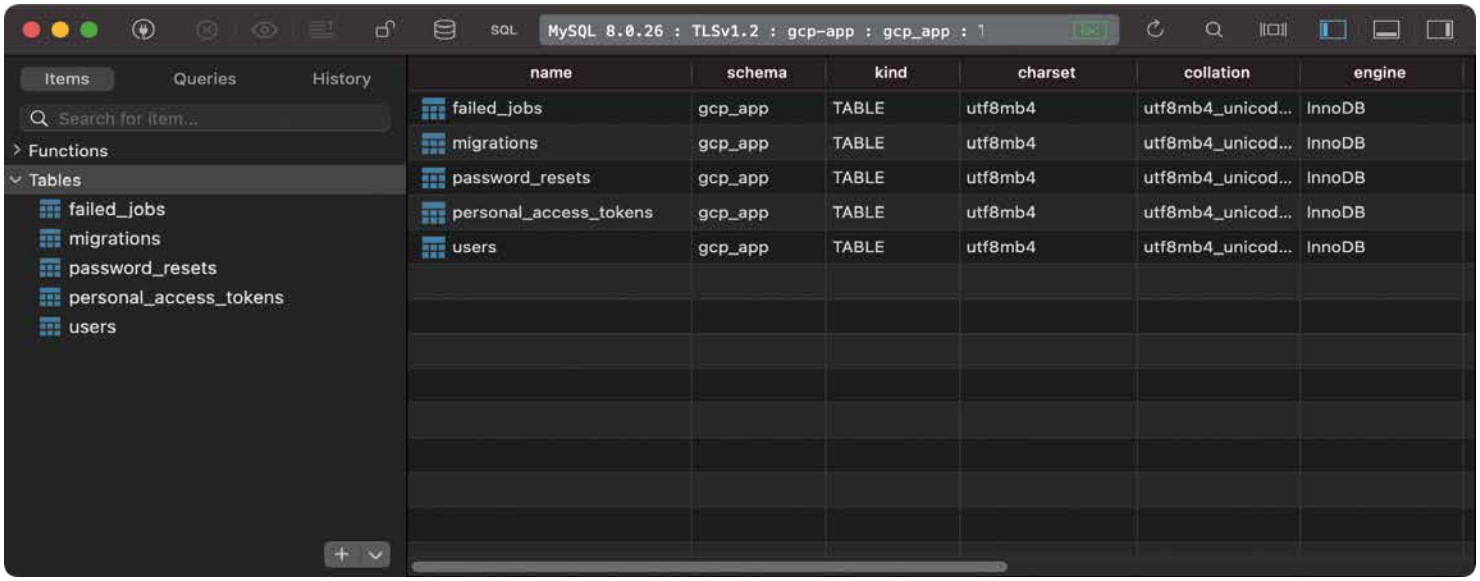


Figure 2: Open database connection

- User: The database user
- Password: The user password
- Database: The name of the database

Once you're done filling in all the necessary fields, click the **Connect** button. **Figure 2** shows the database tables.

You might be wondering where the tables came from. This is the result of creating and running the app in the first part of this series. You ran a command to migrate the database and create all the tables that shipped with Laravel.

The command to run is:

```
sail artisan migrate:fresh
```

Now that you've successfully connected your app to a fresh copy of a MySQL database, let's build a page to manage code editors that you want to keep track of in your database.

Step 2: Build the Coding Editors' Management Page

Let's build a simple page to manage a list of coding editors. The goal is to allow you to add a new coding editor, with some details, and view the list of all editors you're adding.

Step 2.1: Install and Configure Tailwind CSS

Start by installing and configuring **Tailwind CSS** (https://tailwindcss.com/) in the Laravel project. It's a booming CSS framework that I'll use to style the page you're building in this section. You can check out their online guide on how to install and use it inside a Laravel project (https://tailwindcss.com/docs/guides/laravel).

Another option is to make use of Laravel-frontend-presets/tailwindcss package, a Laravel front-end scaffolding preset for Tailwind CSS (https://github.com/laravel-frontend-presets/tailwindcss).

Step 2.2: Create the Editors' Blade Page

Now that you've installed and configured Tailwind CSS, let's create your first Blade (https://laravel.com/docs/8.x/blade) view to manage the editors.

Listing 1: .env Database Settings

```
    mysql:
        image: 'mysql:8.0'
        ports:
            - '${FORWARD_DB_PORT:-3306}:3306'
        environment:
            MYSQL_ROOT_PASSWORD: '${DB_PASSWORD}'
            MYSQL_DATABASE: '${DB_DATABASE}'
            MYSQL_USER: '${DB_USERNAME}'
            MYSQL_PASSWORD: '${DB_PASSWORD}'
            MYSQL_ALLOW_EMPTY_PASSWORD: 'yes'
        volumes:
            - 'sailmysql:/var/lib/mysql'
        networks:
            - sail
        healthcheck:
          test: ["CMD", "mysqladmin", "ping", "-p${DB_PASSWORD}"]
          retries: 3
          timeout: 5s
```

Locate the **\resources\views** folder and create the **editors.blade.php** file. Inside this file, paste the content you can find in this public GitHub Gist: Editors View (https://gist.github.com/bhaidar/e9c4516074a2346f0ce226ce92003cfc).

The view is straightforward. It consists of an HTML Form to allow the user to add a new editor, and a table underneath to show all stored editors in the database. This will do the job to demonstrate a Laravel database connection.

Step 2.3: Add a Route for the New View

To access the new view in the browser, let's add a new route to point to this new view. Locate the \routes\web.php file and append the following two routes:

```
Route::get(
  '/editors',  [
  \App\Http\Controllers\EditorController::class,
'index'
])->name('editors.index');

Route::post(
  '/editors', [
```

```
  \App\Http\Controllers\EditorController::class,
    'store'
]);
```

The first route allows users to access the view (GET) and has a route name of **editors.index**. The second route, on the other hand, allows executing a POST request to create a new editor record in the database.

Listing 2: Store() method

```
public function store(Request $request)
    {
        $request->validate([
              'name' => 'required',
              'company' => 'required',
              'operating_system' => 'required',
              'license' => 'required',
        ]);

        Editor::create($request->all());

        return redirect()->route('editors.index')
            ->with('success', 'Editor created successfully.');
    }
```

Listing 3: Database migration

```
class CreateEditorsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('editors', function (Blueprint $table) {
            $table->id();
            $table->string('name', 255);
            $table->string('company', 500);
            $table->string('operating_system', 500);
            $table->string('license', 255);
            $table->timestamp('created_at')->useCurrent();
            $table->timestamp('updated_at')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('editors');
    }
}
```

Listing 4: Editor Model class

```
class Editor extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'company',
        'operating_system',
        'license',
        'created_at'
    ];
}
```

The routes use the **EditorController** that you haven't created yet. Let's run a new Artisan command to create this controller:

```
sail artisan make:controller EditorController
```

This command creates a new controller under the **\App\Http\Controllers** folder.

The **index()** action retrieves all stored editors in the database and returns the editors view together with the data to display.

```
public function index()
{
   $editors = Editor::all();
   return view('editors', compact('editors'));
 }
```

The **store()** action takes care of storing a new editor record in the database. It validates the POST request to make sure that all required fields are there. Then, it creates a new editor record in the database. Finally, it redirects the user to the editors.index route (you have defined this inside \routes\web.php). **Listing 2** shows the store() action source code entirely.

Step 2.4: Create a Laravel Migration

To store information about coding editors, you need to create a corresponding database table. In Laravel, you need to create a new database migration and run it against the connected database. Welcome to Artisan Console!

Laravel ships with Artisan (https://laravel.com/docs/8.x/artisan), a command-line interface (CLI), that offers many commands to create functionality in the application. It's located at the root of the project folder and can be called like any other CLI on your computer.

To create a new Laravel database migration, run the following command:

```
sail artisan make:model Editor -m
```

This command makes a new Model (https://laravel.com/docs/8.x/eloquent) class together with a database migration file (-m option).

> Laravel's make:model Artisan command can generate a model, controller, migration, and factory in one single command. You can run the options using -mcf.

Locate the **\database\migrations** folder and open the new migration PHP file that you just created. Replace the content of this file with the content showing in **Listing 3**.

The core of every migration file is the **up()** method.

```
Schema::create('editors',
  function (Blueprint $table) {
    $table->id();
```

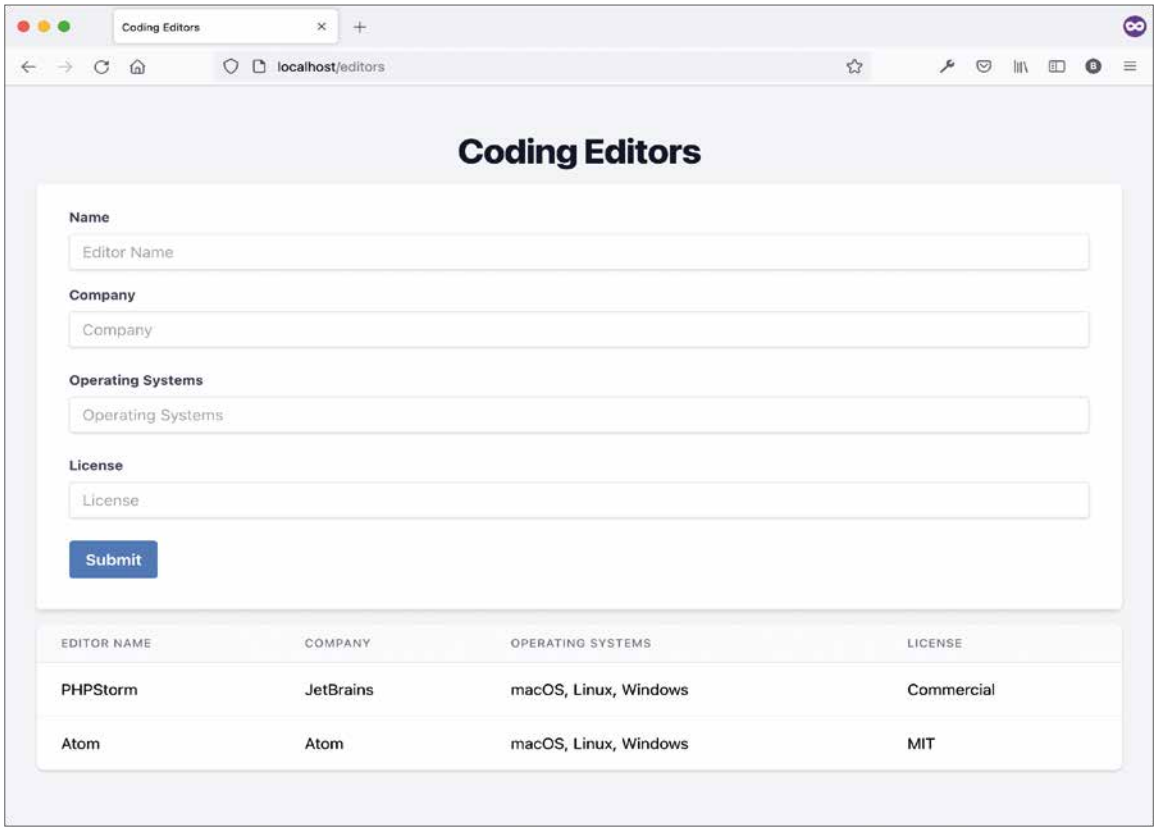


Figure 3: Editors' view in the browser

```
    $table->string('name', 255);
    $table->string('company', 500);
    $table->string('operating_system', 500);
    $table->string('license', 255);
    $table->timestamp('created_at')->useCurrent();
    $table->timestamp('updated_at')->nullable();
});
```

Inside the up() method, you're creating the **editors** table and specifying the columns that should go under it. **Listing 4** shows the Editor model class.

I've added the **$fillable** property to whitelist the columns that are available for the mass assignment. This comes in later when creating and storing editors in the database.

You can read more about $fillable and $guarded in Laravel by checking this awesome and brief introduction on the topic: https://thomasventurini.com/articles/fillable-vs-guarded-on-laravel-models/.

Now that the migration and model are both ready, let's run the migration to create the new table in the database. Run the following command:

```
sail artisan migrate
```

You can verify that this command has created the table by checking your database.

Step 2.5: Run the App

The final step is to run the app and start using the editors' route to store a few coding editors in the database and make sure the database connection is up and running. **Figure 3** shows the Editors' view in the browser.

That's it! Now that you've successfully connected your app to a local database, let's explore Google Cloud SQL and configure the app to use one.

Before moving on, make sure you commit and push your change onto GitHub. Keep in mind that this action triggers the GCP Cloud Build Workflow to deploy a new version of your Laravel app.

Create and Use a Google Cloud SQL Database

Google Cloud SQL is a fully managed relational database that supports MySQL (https://www.mysql.com/), PostgreSQL (https://www.postgresql.org/), and Microsoft SQL Server (https://www.microsoft.com/en-us/sql-server/sql-server-downloads).

> Google Cloud SQL is a fully managed relational database that supports MySQL, PostgreSQL, and Microsoft SQL Server

You can read the full documentation on Google Cloud SQL here: https://cloud.google.com/sql.

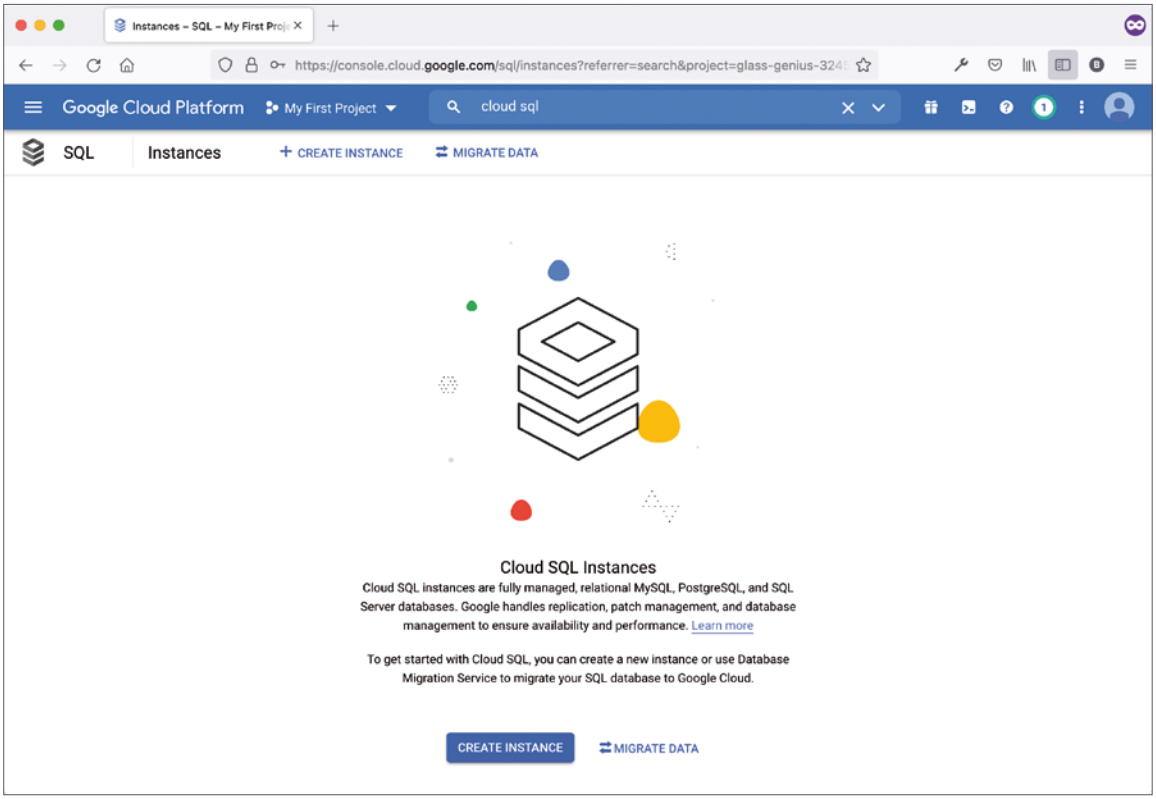


Figure 4: Google Cloud SQL

In this section, you're going to create your first Cloud SQL and connect it from your Laravel app running on the GAE (Google App Engine).

Log into your account at https://console.cloud.google.com/ and navigate to the Cloud SQL section by selecting it from the left-side menu. **Figure 4** shows where to locate the Cloud SQL on the main GCP menu.

The GCP (Google Cloud Platform) takes you through a few steps to help you easily create a new instance. Let's start!

Step 1: Create a MySQL Instance

Locate and click the **CREATE INSTANCE** button. Follow the steps to create your first Cloud SQL instance.

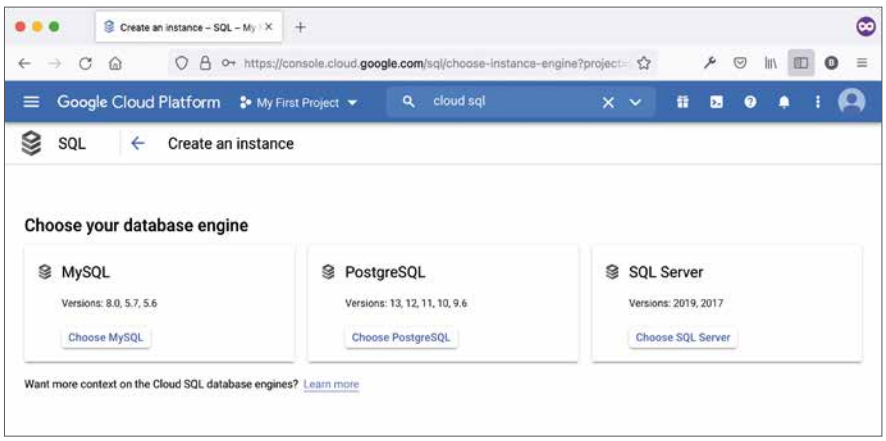


Figure 5: GCP database engine offerings

The next step is to select which database engine you're going to create. For this series, stick with a MySQL database. **Figure 5** shows the database engine offerings by GCP.

Select the **Choose MySQL** button. Next, GCP prompts you to fill in the configuration details that GCP needs to create your MySQL instance. **Figure 6** shows the MySQL instance configuration settings.

At a minimum, you need to input the following fields:

- Instance ID
- Password (for the root MySQL instance user). Make sure you remember this password as you'll need it later.
- Database version. I'll stick with MySQL 5.7 for now.
- Region (preferably the same region you picked for the GAE app)
- Zonal availability (either single or multiple zones, depending on your requirements and needs)

The rest of the fields are optional. Look at them in case you want to change anything.

Click the **CREATE INSTANCE** button. GCP starts creating the instance and directs you to the Cloud SQL Dashboard upon completion. **Figure 7** shows the Cloud SQL Dashboard.

From here, you'll start configuring the MySQL instance and preparing it for connection from your Laravel app.

Step 2: Create a MySQL Database

On the **gcp-app-database** dashboard, locate and click the **Databases** on the left-side menu. This page lists all the databases you create under the MySQL Instance.

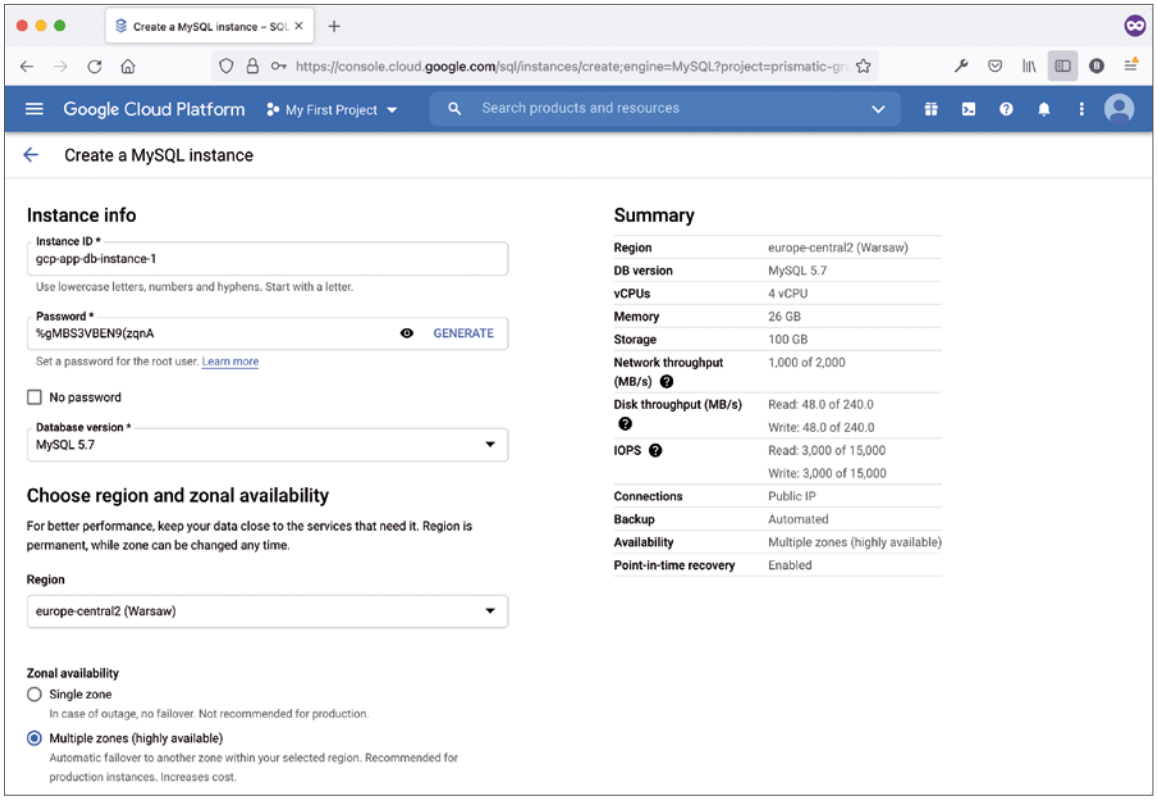


Figure 6: MySQL instance configuration settings

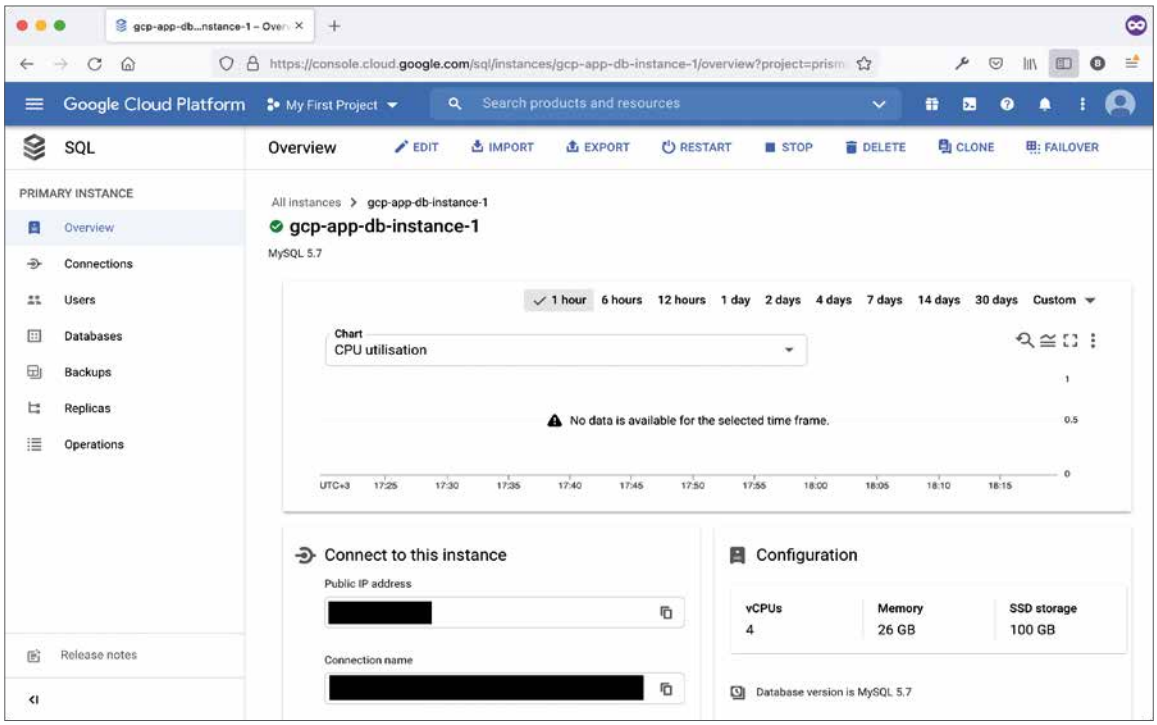


Figure 7: Cloud SQL dashboard

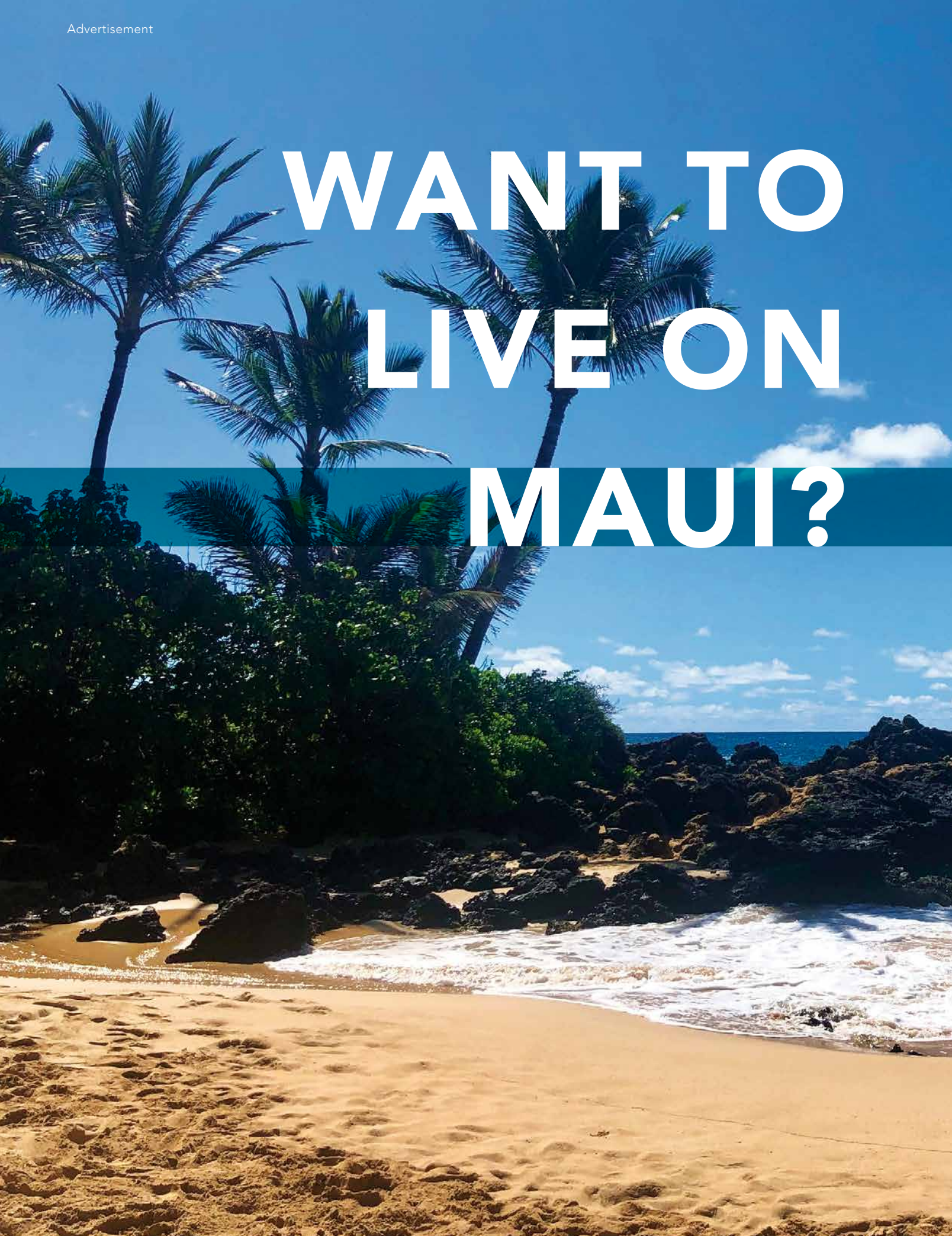
Click the **CREATE DATABASE** button to create a database for the Laravel app. **Figure 8** shows the create database form.

Provide a name for the new database and click the **CREATE** button. Just a few seconds later, you'll see the new database listed under the current instance list of databases.

Locate and click the **Connections** on the left-side menu. This page lists all the networking configurations that govern your database instance.

For now, keep selecting the **Public API** option. It allows you to connect to your database by using the Cloud SQL Proxy.

Advertisement



IF YOU CAN WORK FROM HOME, WHY NOT MAKE PARADISE YOUR HOME?

The world has changed. Millions of people are working from home, and for many, that will continue way past the current crisis. Which begs the question: If you can work from home, then why not make your home in one of the world's premiere destinations and most desirable living areas?

The island of Maui in Hawai'i is not just a fun place to visit for a short vacation, but it is uniquely situated as a place to live. It offers great infrastructure and a wide range of things to do, not to mention a very high quality of life.

We have teamed up with *CODE Magazine* and Markus Egger to provide you information about living in Maui. Markus has been calling Maui his home for quite some time, so he can share his own experience of living in Maui and working from Maui in an industry that requires great infrastructure.

For more information, and a list of available homes, visit **www.Live-On-Maui.com**

Steve and Carol Olsen
Maui, Hawai'i

MAUI NO KA OI!

This loosely translates to "Maui is the best". As someone who has been calling Maui home for a while now, I can wholeheartedly confirm this. After having travelled the world, and after having lived in a variety of places, I find Maui to be truly unique.

Most people know Maui is a place to go for a week on vacation. And that is certainly great and very enjoyable. However, Maui is so much more! To me, Maui is the perfect mix that makes me feel like I am living on a tropical island yet being a developed place with great infrastructure and great quality of life. A lot of this is true for the Hawaiian Islands in general. But while Oahu (with its capital of Honolulu) is essentially a big city with a lot of people that always reminds me of Southern California, and while islands like Kauai or the Big Island of Hawai'i are a bit too "back to the roots" for me, Maui is just perfect. You can enjoy a great day at the beach or in nature, or you can go to a nice restaurant, the movies, or a concert. It's the quality of life provided by a modern place in the Western world, paired with a tropical island paradise.

Maui has many unique advantages. There is no hurricane season and no real rainy season. The weather is nice year-round, especially on the south-side of the island. There are no dangerous animals. Not even mosquitoes. How does 82-degree weather on a nice beach with a Mai Tai or Pina Colada sound? That's Maui for you!

As someone who works in the tech industry, good infrastructure is important to me. After all, I need to work as productively from Maui as I do when I am on the "mainland" (which is what we call the *continental US* here in Hawai'i). I have a 300Mbit internet connection going to my home, and it is inexpensive. My connection to other parts of the world is better and less expensive here than it is on most main-land locations. We have the same stores and supermarkets as everywhere else in the US. Schools are decent. The same is true for healthcare. Flight connections are great, and it is not at all difficult to travel from and to Maui.

I live on the south-side of the island in an area called "Kihei". Especially the southern parts of Kihei, known as "Wailea" and "Makena", are the areas I truly recommend to anyone (although there are other places worth considering also). I like this area for its great quality of housing, low crime, great weather, and the world's greatest beaches. I enjoy playing a round of golf or going to a great restaurant with friends. When I want to feel like I'm on vacation for an hour or two, I swing by one of the hotels for a snack at the pool bar. When I feel like exercising, I ride my bike along the ocean or go for a hike into the jungle or across lava fields.

As we have been going through the COVID-19 crisis, it has become more and more clear how great a location Maui is. For one, the warm weather and outdoor living have kept the COVID-19 numbers low, and the quality of life high. And while nobody wants to be hospitalized, it has been nice to know that we have better healthcare here than other tropical locations. (Essentially the same healthcare as anywhere else in the US.) While we also had to deal with a lockdown, and many restaurants and hotels have been closed, we have several places that are not just open, but since everything happens outdoors, many are perfectly safe to visit. I really can't think of a better place to weather this pandemic than Maui.

So yes: Maui No Ka Oi! To me, there isn't another place that even comes close. I have a long list of other places I enjoy visiting that are awesome too. Do I want to go to Bora Bora, Singapore, or many other great locations? Sure, I do! But what do you do in Bora Bora after two weeks? Maui on the other hand is a great place to set up a life and stay for good.

I would love to see you on Maui in the future. Maybe we can share one of those Mai Tais on the beach. I recommend talking to Carol Olsen, who has been helping me with all my real estate needs in South Maui. Moving to Maui has been the best decision of my life. I am sure you would enjoy it too!

Markus Egger
Publisher, *CODE Magazine*

MAUI PROPERTIES

Wailea Homes

Wailea Condos

Wailea Land

Kihei Homes

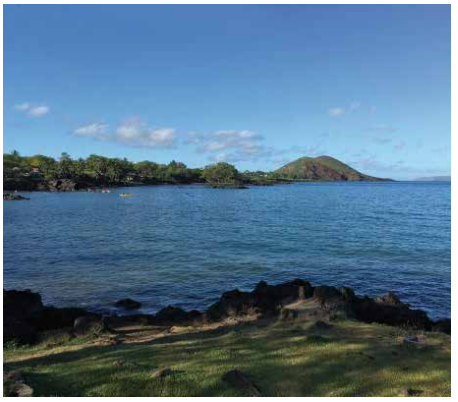
Kihei Condos

Kihei Land

Makena Home

Makena Condo

Makena Land



Step 3: Use Cloud SQL Auth Proxy to Connect to the Database

The Cloud SQL Auth proxy (https://cloud.google.com/sql/docs/mysql/sql-proxy) provides secure access to your Cloud SQL instances without the need for Authorized networks or for configuring SSL. Download the Cloud SQL Auth Proxy version that best fits in your environment by checking this resource: https://cloud.google.com/sql/docs/mysql/sql-proxy#install

You have multiple ways to use the Cloud SQL Auth Proxy. You can start the Cloud SQL Auth proxy using TCP sockets, Unix sockets, or the Cloud SQL Auth proxy Docker image. You can read about how to use the Cloud SQL Auth Proxy here: https://cloud.google.com/sql/docs/mysql/connect-admin-proxy#tcp-sockets

For this series, I'll be using the TCP sockets connection. Use the following command to connect to the Cloud SQL instance:

```
./cloud_sql_proxy \
    -instances=INSTANCE_CONNECTION_NAME=tcp:3306
```

INSTANCE_CONNECTION_NAME is the connection name that GCP provides, and you can locate it in **Figure 7** under the **Connection name** field.

To start the Cloud SQL Auth Proxy, run the following command:

```
./cloud_sql_proxy \
    -instances=INSTANCE_CONNECTION_NAME=tcp:3306
```

Replace the INSTANCE_CONNECTION_NAME with the real connection name.

Figure 9 shows the Cloud SQL Auth Proxy connected and ready to establish a database connection to any database under the currently connected instance.

With TCP connections, the Cloud SQL Auth proxy listens on localhost (127.0.0.1) by default. So when you specify tcp:PORT_NUMBER for an instance, the local connection is at 127.0.0.1:PORT_NUMBER. **Figure 10** shows how to connect to the db_1 using TablePlus (https://tableplus.com/).

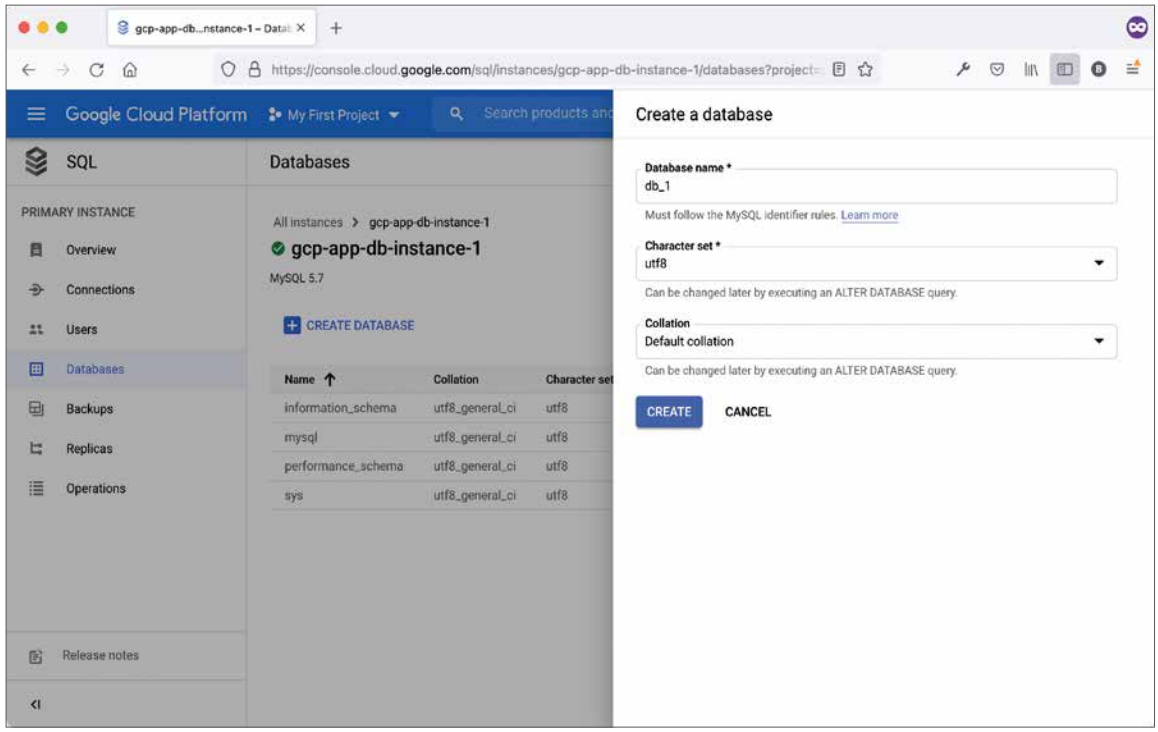


Figure 8: Create a new MySQL database.

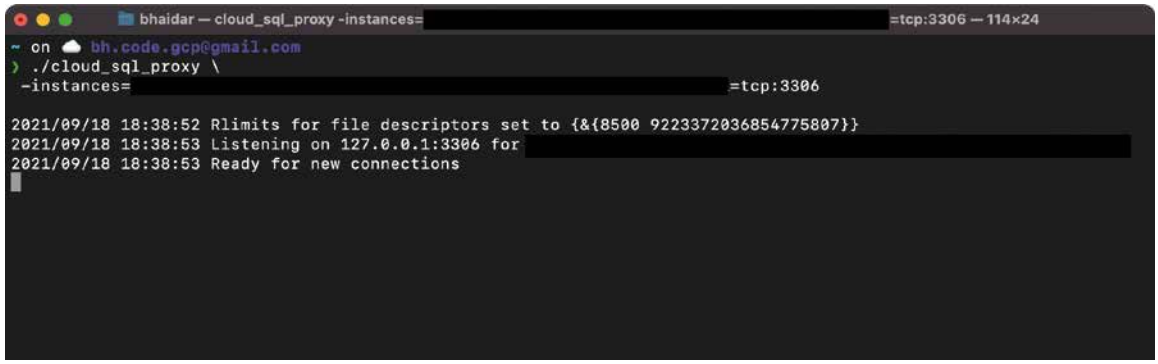


Figure 9: Cloud SQL Auth Proxy connection

I've highlighted the important fields that need your attention:

- Name: The name of the connection
- Host: The IP address of the server hosting the MySQL database. In this case, it's 127.0.0.1.
- User: The database user. In this case, you're using the root user.
- Password: The user password. The root password that you previously created for the MySQL instance.
- Database: The name of the database. In this case, it's db_1.

Click the **Connect** button to successfully connect to the database. The database is still empty and you'll fill it with Laravel tables in the next section.

Step 4: Run Laravel Migrations on the New Database

In Steps 2.4 and 2.5, you created a database migration and pushed the code to GitHub. By pushing the code, the GCP Cloud Build Workflow runs and deploys a new version of the app. This means that your app on the GAE is now up to date, with the editors' view up and running.

Before you test your view on GAE, let's run the Laravel migrations on the cloud database. While the Cloud SQL Auth Proxy is running, switch to the app source code and apply the following changes.

Locate and open the **.env** file and update the database section as follows:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_1
DB_USERNAME=root
DB_PASSWORD=
```

Make sure you replace the DB_HOST with 127.0.0.1. Also replace the DB_DATABASE with db_1, the DB_USERNAME with root, and, finally, set the DB_PASSWORD for the root user.

On a terminal window, run the following command to refresh the app configurations:

```
php artisan config:cache
```

This command clears the old configurations and caches the new ones.

The app, when it runs, is now connected to the Cloud database. Hence, to run the Laravel migrations, you just need to run the following command:

```
php artisan migrate:fresh
```

Notice the use of **php artisan** rather than **sail artisan**. You use the **sail** command only when interacting with the Sail environment locally.

Switch back to TablePlus to see all the tables there. **Figure 11** shows all the tables running a fresh Laravel migration.

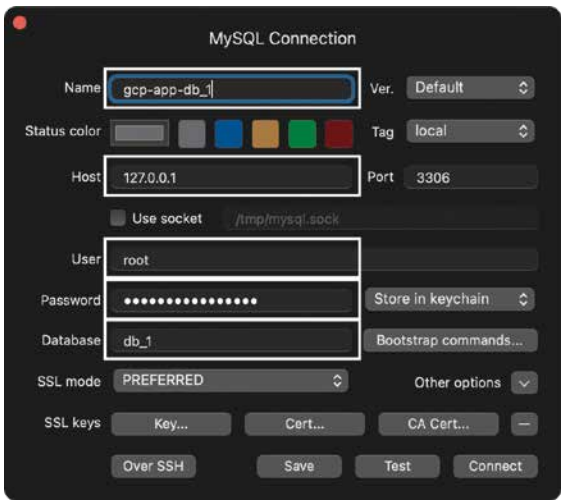


Figure 10: Connect to db_1 via Cloud SQL Auth Proxy

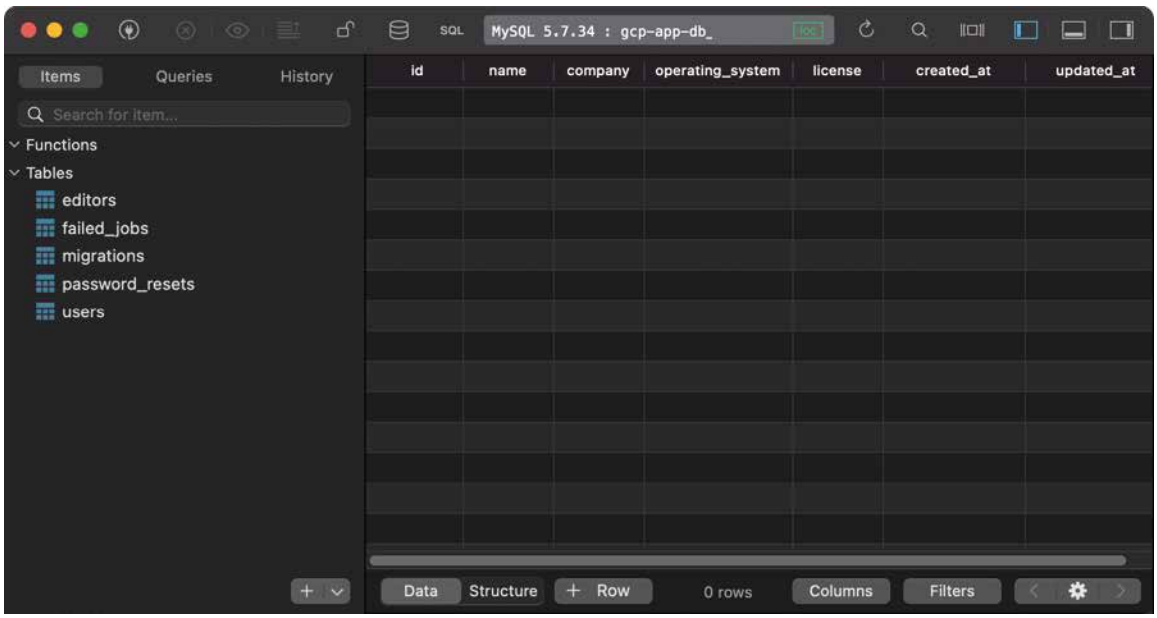


Figure 11: Tables created by running a fresh Laravel migration

You have successfully prepared the database and are ready to accept new connections.

Step 5: Enable GCP APIs and Permissions

Before you can connect to the cloud database from within the GAE, you need to enable a few libraries and add some permissions.

Locate the following APIs and enable them in this order:

- Cloud SQL Admin API
- Google App Engine Flexible Environment

> In Google Cloud, before using a service, make sure to enable its related APIs and Libraries.

To enable any API or Library on the GCP, on the left-side menu, click the **APIs and Services** menu item. Then, once you're on the APIs and Service page, click the **Library** menu item. Search for any API and enable it by clicking the **ENABLE** button.

In addition to enabling the two APIs, you need to add the role of Cloud SQL Client for the GAE Service Account under **IAM and Admin** section.

On the left-side menu, locate and click the IAM and Admin menu item. Click the pencil icon to edit the account that ends with **@appspot.gserviceaccount.com** and that has the name of **App Engine default service account**. **Figure 12** shows how to add the Cloud SQL Client role.

Now that you've configured all APIs and libraries, let's connect the Laravel app that's running on GAE to the cloud database.

Step 6: Configure Laravel App on GAE to Use the Cloud Database

It's time to configure your app running inside GAE to connect to this cloud database. Start by locating and opening the **\app.yaml** file at the root folder of the app.

Open the file and append the following settings under the **env_variables** section:

```
  DB_DATABASE: db_1
  DB_USERNAME: root
  DB_PASSWORD:
  DB_SOCKET: '/cloudsql/INSTANCE_CONNECTION_NAME'
```

Replace the INSTANCE_CONNECTION_NAME with the real connection name. Then, add a new section to the **\app.yaml** file:

```
beta_settings:
  cloud_sql_instances: 'INSTANCE_CONNECTION_NAME'
```

This enables GAE to establish a Unix domain socket with the cloud database. You can read more about connecting to the Cloud database from GAE here https://cloud.google.com/sql/docs/mysql/connect-app-engine-flexible#php.

Commit and push your changes on GitHub. This triggers the Google Cloud Build workflow to run and deploy a new version of the app. Wait until the GCP deploys your app, then access the editors' view by following this URL: https://prismatic-grail-323920.appspot.com/editors.

This URL opens the coding editors' view. Start adding a few coding editors to try out the database connection and make

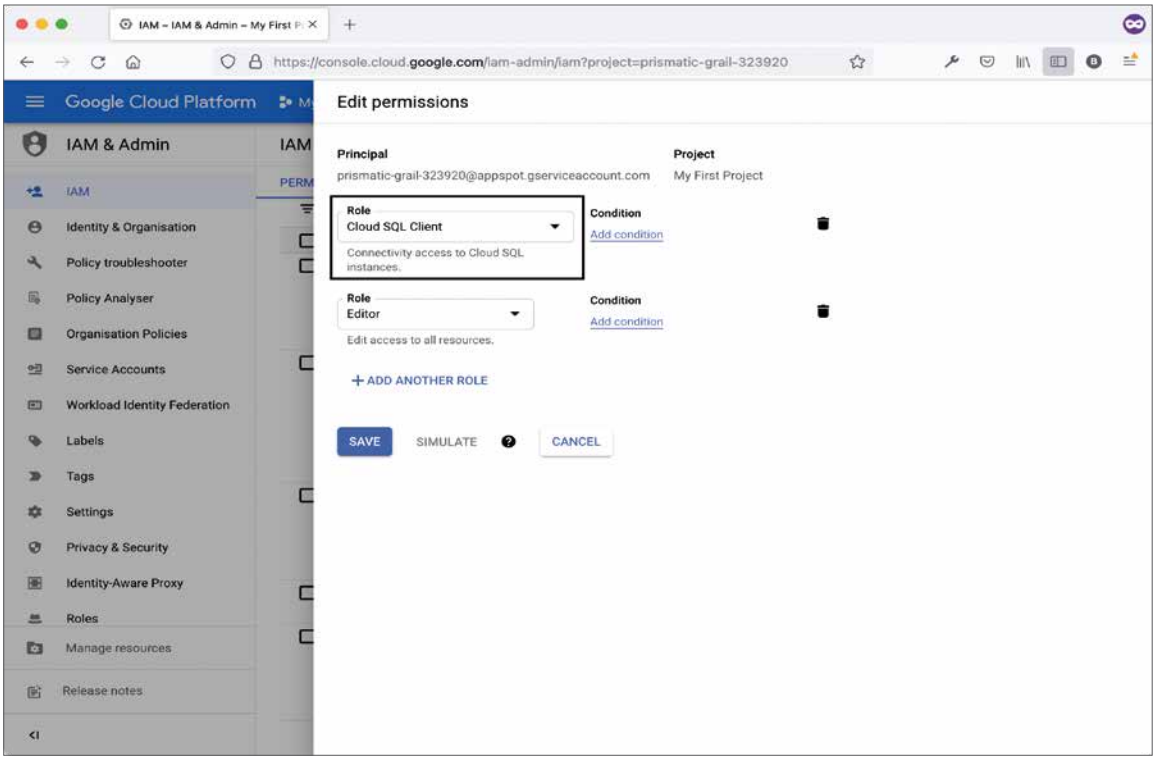


Figure 12: Adding the Cloud SQL Client role on the App Engine default user

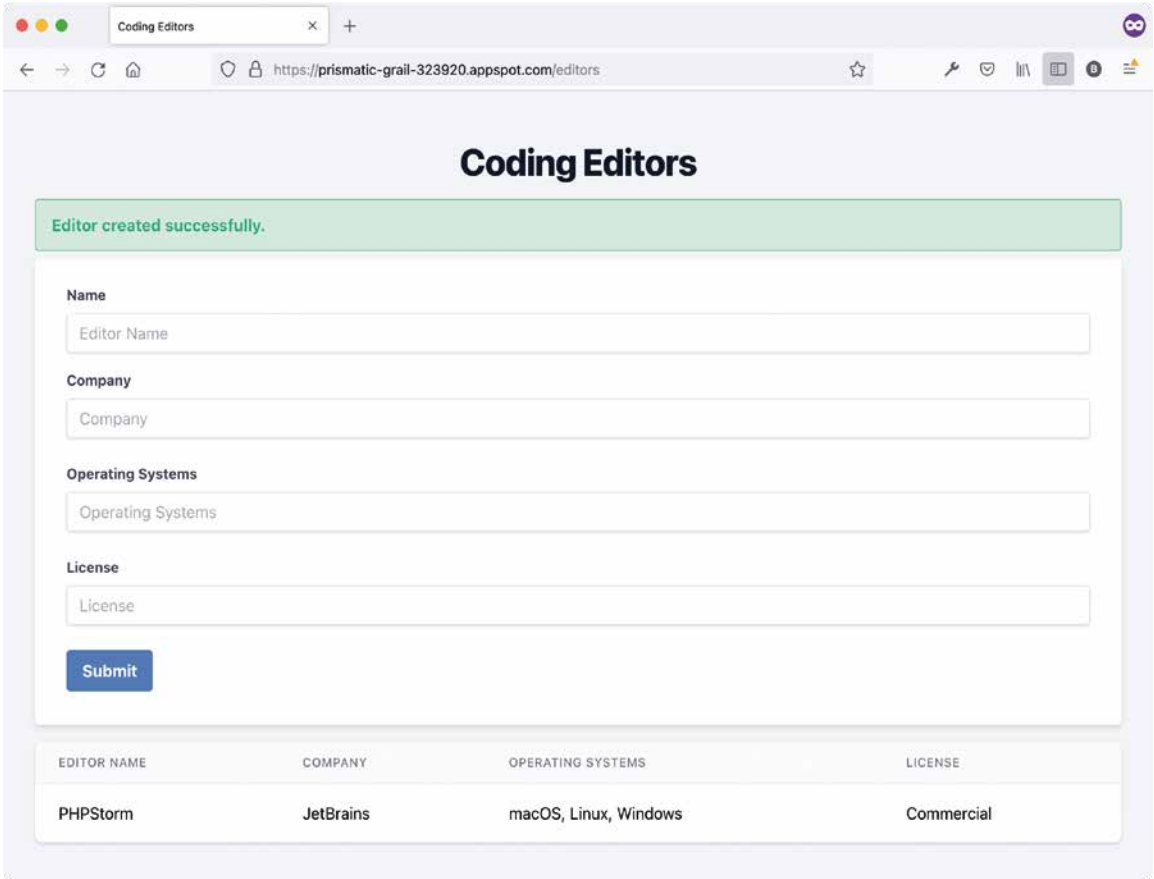


Figure 13: Editors' view up and running on GCP

sure all is working smoothly. **Figure 13** shows the Editors' view up and running on GCP.

You have successfully connected your Laravel app to the Google Cloud SQL database. Let's move on and enhance the Google Cloud Build workflow.

Run Laravel Migrations inside the Cloud Build Workflow

When you're deploying your app to GAE, there's almost no easy way to access the underlying Docker container and run your migrations. You need to automate this task as part of the Cloud Build workflow.

One way to automate running Laravel migrations inside a Cloud Build workflow is the following:

- Add a new controller endpoint in your app that can run the Artisan migration command. You need to secure this endpoint by authenticating the request and making sure it's coming solely from GCP. There are ways to do so, as you will see later.
- Add a Cloud Build step to issue a curl (https://gist.github.com/joyrexus/85bf6b02979d8a7b0308) POST request to the endpoint from within the Cloud Build workflow.

Step 1: Add a Controller Endpoint to Run the Migrations

Let's start by adding a new invokable controller in Laravel by running the following command:

Listing 5: SetupController __invoke() method

```
public function __invoke(Request $request):
 \Illuminate\Http\Response
    {
        try {

            Log::debug('Starting: Run database migration');

            // run the migration
            Artisan::call('migrate',
               [
                '--force' => true
               ]
            );

            Log::debug('Finished: Run database migration');

        } catch (\Exception $e)
        {
            // log the error
            Log::error($e);

            return response('not ok', 500);
        }

        return response('ok', 200);
    }
```

```
sail artisan make:controller SetupController \
    --invokable
```

An invokable controller is a single action controller that contains an **__invoke** method to perform a single task. **Listing 5** shows the __invoke() function implementation.

The function is simple. It calls the migrate command using the **Artisan::call()** function call. It also does some logging to trace whether this task runs or fails.

The next step is to add a new route inside the file **\routes\web.php** as follows:

```
Route::post(
  '/setup/IXaOonJ3B7',
  '\App\Http\Controllers\SetupController'
);
```

I'm adding a random string suffix to the **/setup/** URL, trying to make it difficult to guess this route path. One final step is to locate and open the **\app\Http\Middleware\VerifyCsrfToken.php** file. Then, make sure to enlist the /setup/ URL inside the $except array as follows:

```
protected $except = [
        'setup/IXaOonJ3B7'
    ];
```

This way, Laravel won't do a CSRF token verification (https://laravel.com/docs/8.x/csrf) when the /setup/ URL GCP requests it.

Step 2: Amend Google Cloud Build to Run Migrations

Switch to **\ci\cloudbuild.yaml** and append a new Cloud Build step to invoke the /setup/ URL from within the Build workflow. **Listing 6** shows the build step to invoke the /setup/ URL.

Listing 6: Invoke /setup/ inside Google Build file

```
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: "bash"
  args:
    - "-c"
    - |
      RESPONSE=$(curl -o /dev/null -s -w "%{http_code}" \
                  -d "" -X POST $_APP_BASE_URL)
      if [ "200" != "$$RESPONSE" ];
      then
        echo "FAIL: migrations failed"
        exit 1;
      else
        echo "PASS: migrations ran successfully"
      fi
```

The build step uses a container of the **gcr.io/cloud-builders/gcloud** Docker image to run a curl command on a new bash shell. To learn more about Google Cloud Build Steps, check this resource https://cloud.google.com/build/docs/build-config-file-schema.

The build step issues a POST curl request to a URL represented by **$_APP_BASE_URL**. The Google Cloud Build substitutes this variable with an actual value when it runs the trigger. The value of this variable shall be the full app /setup/ URL. You can learn more about Google Cloud Build substitution here: https://cloud.google.com/build/docs/configuring-builds/substitute-variable-values.

Step 3: Amend the Cloud Build Trigger to Pass Over the _APP_BASE_URL

Visit the list of triggers under Google Cloud Build. Locate the **deploy-main-branch** trigger and click to edit. **Figure 14** shows how to edit a Google Cloud Build trigger.

Once on the Edit Trigger page, scroll to the bottom, locate, and click the **ADD VARIABLE** button. This prompts you to enter a variable name and value. Variable names should start with an underscore. To use this same variable inside the Google Cloud Build workflow, you need to prefix it with a **$** sign. At run time, when the trigger runs, GCP substitutes the variable inside the Build workflow with the value you've assigned on the trigger definition. **Figure 15** shows the _APP_BASE_URL variable together with its value.

Save the trigger and you're ready to go!!

Step 4: Run and Test the Trigger

Before running the trigger, let's make a new migration, for example, to add a new **description** column on the **editors** table.

Run the following command to generate a new migration file:

```
sail artisan make:migration \
     add_description_column_on_editors_table
```

Listing 7 shows the entire migration file.

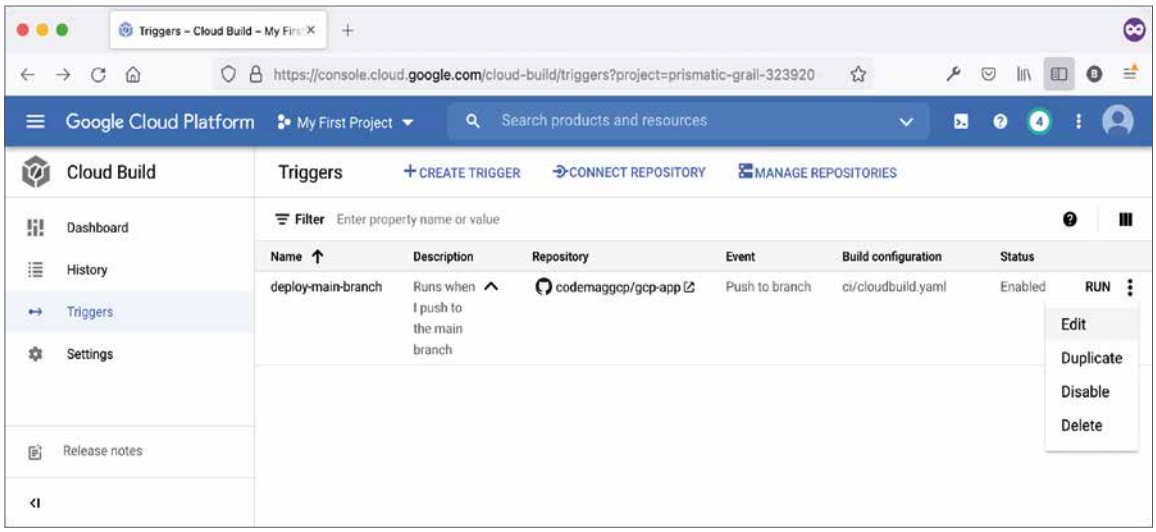


Figure 14: Edit the Google Cloud Build trigger.

Listing 7: Database migration file

```
class AddColumnDescriptionOnEditorsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('editors', function($table) {
            $table->string('description', 255)->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('editors', function($table) {
            $table->dropColumn('description');
        });
    }
}
```

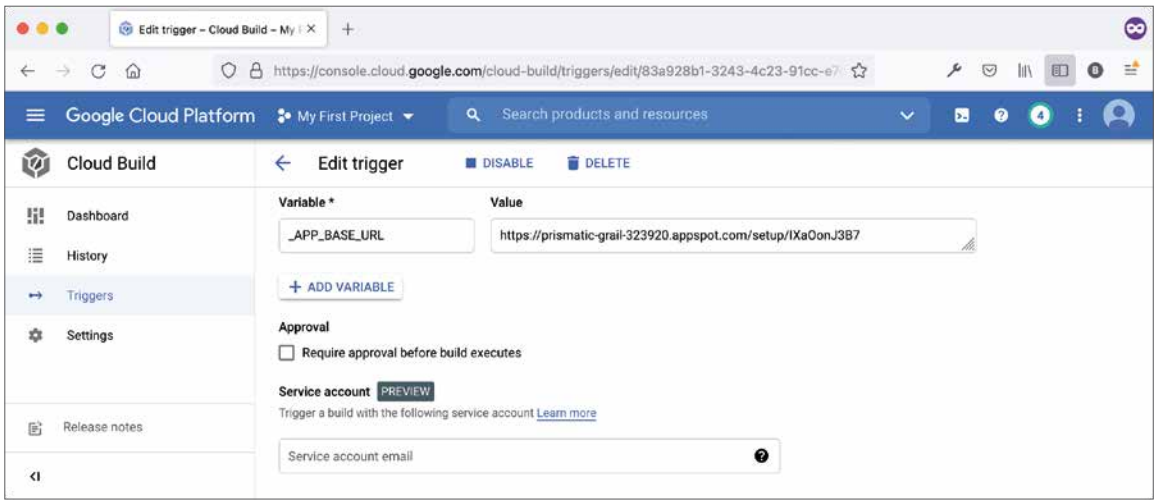


Figure 15: Adding a new trigger variable

The migration, when run, adds a new column on the editors table inside the database.

Save your work by adding all the new changes to Git and committing them to GitHub. This, in turn, triggers GCP to run the associated Google Cloud Build workflow.

Eventually, a new version of the app will be deployed on GCP. However, this time, the trigger will also POST to the /setup/ URL to run the Laravel migrations as part of running the Build workflow.

You can check the database to make sure the Build workflow runs the Laravel migration and accordingly adds a new column to the editors table.

So far, the /setup/ URL isn't authenticating the request coming and ensuring that it's coming from GCP only. In the next part of this series, I'll explore one option to secure and authenticate your requests by using Google Secret Manager (https://cloud.google.com/secret-manager).

Conclusion

This article was a continuation of the previous one by connecting your Laravel app to a local database. The next step is to create a Google Cloud database and connect the app to it when running inside GAE. Finally, you enhanced the Google Cloud Build workflow to run the Laravel migrations as part of the Build workflow itself.

Next time I see you, I'll strengthen running the Laravel migration by implementing an authentication layer with Google Secret Manager.

In addition, deployments can go wrong sometimes and so you should have a backup plan. As you will see in the next article, you can take a backup snapshot of the Cloud database every time you run the Build workflow. If anything goes wrong, you can revert to an old backup of the database.

And much more... See you then!

Bilal Haidar

Working with Apache Kafka in ASP.NET 6 Core

Today's enterprises need reliable, scalable, and high-performant distributed messaging systems for data exchange in real-time. There are quite a few messaging systems out there, and Apache Kafka is one of them. It's an open source, and versatile stream-processing software that's a high-throughput, low-latency messaging system for distributed applications, and it's written in Java and Scala.

Joydip Kanjilal
joydipkanjilal@yahoo.com

Joydip Kanjilal is an MVP (2007-2012), software architect, author, and speaker with more than 20 years of experience. He has more than 16 years of experience in Microsoft .NET and its related technologies. Joydip has authored eight books, more than 500 articles, and has reviewed more than a dozen books.

This article provides a deep dive on how to work with Apache Kafka in ASP.NET 6 Core.

If you're to work with the code examples discussed in this article, you should have the following installed in your system:

- Visual Studio 2022
- .NET 6.0
- ASP.NET 6.0 Runtime
- Apache Kafka
- Java Runtime Environment (JRE)
- 7-zip

You can download and install 7-zip from here: https://www.7-zip.org/download.html.

You can download JRE from here: https://www.java.com/en/download/.

If you don't already have Visual Studio 2022 installed in your computer, you can download it from here: https://visualstudio.microsoft.com/downloads/.

You can download Apache Kafka from here: https://kafka.apache.org/downloads.

> Take advantage of Apache Kafka for high performance, scalable, and reliable messaging in real-time.

Introduction to Apache Kafka

Streaming data refers to data constantly produced by hundreds of data sources, which often transmits the data records concurrently. A streaming platform must manage this continual influx of data while still processing it sequentially and progressively.

Kafka is a publish/subscribe messaging platform with built-in support for replication, partitioning, fault tolerance, and better throughput. It's an excellent choice for applications that need large scale data processing. Kafka is mainly used to build real-time streaming data pipelines. Kafka incorporates fault-tolerant storage and stream processing capabilities to allow for the storage and analysis of historical and real-time data.

Here's the list of Apache Kafka features:

- It can publish and subscribe streams of data.
- It's capable of handling a vast number of read/write operations per second.
- It can persist data for a particular period.
- It has the ability to grow elastically with zero downtime.
- It offers support for replication, partitioning, and fault-tolerance.

Why Should You Use Apache Kafka?

Scalability. Apache Kafka is highly scalable. It supports high-performance sequential writes and separates topics into partitions to facilitate highly scalable reads and writes. This helps Kafka to enable multiple producers and consumers to read and write at the same time. Additionally, because Kafka is distributed, you can scale up by adding new nodes to the cluster.

High throughput. Throughput is a measure of the number of messages that arrive at a given point in time. Apache Kafka is capable of handling massive volumes of incoming messages at a high velocity per second (around 10K messages per second or a maximum request size of one million bytes per request, whichever comes first).

High performance. Apache Kafka can deliver messages at high speed and high volumes. It provides high throughput with low latency and high availability.

Highly reliable. Kafka is a fault-tolerant messaging system and is adept at recovering from failures quickly. Kafka can replicate data and handle many subscribers. In Apache Kafka, the messages are durable even after they have been consumed. This enables the Kafka Producer and Kafka Consumer to be available at different times and increases resilience and fault tolerance. Kafka can load balance consumers in the event of a failure. It is more reliable than other messaging services such as RabbitMQ, AMQP, JMS, etc.

Low latency. Latency refers to the amount of time required to process each message. Apache Kafka can provide high throughput with low latency and high availability.

Durability. Kafka messages are highly durable because Kafka stores the messages on the disk, as opposed to in memory.

Kafka vs. Traditional Messaging Systems

Kafka differs from traditional messaging queues in several ways. Kafka retains a message after it has been consumed. Quite the opposite, competitor RabbitMQ deletes messages immediately after they've been consumed.

RabbitMQ pushes messages to consumers and Kafka fetches messages using pulling.

Kafka can be scaled horizontally and traditional messaging queues can scale vertically.

Typical Use Cases

Here are some use cases for Kafka:

- **Messaging:** Kafka acts as a message broker. A message broker enables applications, services, and systems to communicate with one another and exchange information. It can decouple processing from data producers and store, validate, organize, route, and deliver messages to appropriate destinations.
- **Application activity tracking:** Kafka was originally developed to address application activity tracking. You can leverage Kafka to publish all events (user login, user registration, time spent by a logged in user, etc.) that occur in your application to a dedicated Kafka topic. Then you can have consumers subscribe to the topics and process the data for monitoring, analysis, etc.
- **Log aggregation:** You can publish logs to Kafka topics and then aggregate and process them when needed. Kafka can collect logs from various services and make them available to the consumers in a standard format (JSON).
- **Real-time data processing:** Today's applications need data to be processed as soon as it's available. IoT applications also need real-time data processing.
- **Operational metrics:** Kafka can aggregate the statistical data collected from several distributed applications and then produce centralized feeds of operational data.

Components of the Apache Kafka Architecture

The Apache Kafka architecture is comprised of the following components:

- **Kafka Topic:** A Kafka topic defines a channel for the transmission of data. When the producers publish messages to the topics, the consumers read messages from them. A unique name identifies a topic pertaining to a Kafka cluster. There's absolutely no limit to the number of topics you can create in a cluster.
- **Kafka Cluster:** A Kafka cluster comprises one or more servers or Kafka brokers. For high availability, a Kafka cluster typically contains many brokers, each of them having its own partition. Because they're stateless, ZooKeeper (see later in this list) is used to manage the cluster state.
- **Kafka Producer:** A Kafka producer serves as a data source for one or more Kafka topics and is responsible for writing, optimizing, and publishing messages to those topics. A Kafka producer can connect to a Kafka cluster through Zookeeper. Alternatively, it can connect to a Kafka broker directly.
- **Kafka Consumer:** A Kafka consumer consumes data through reading messages on the topics they've subscribed to. Incidentally, each Kafka consumer belongs to a particular consumer group. A Kafka consumer group comprises related consumers who share a common task. Kafka sends messages to the consumers within the group from different partitions of a topic.
- **Kafka ZooKeeper:** A Kafka Zookeeper manages and coordinates the Kafka brokers in a cluster. It also notifies producers and consumers in a Kafka cluster of the existence of new brokers or the failure of brokers.

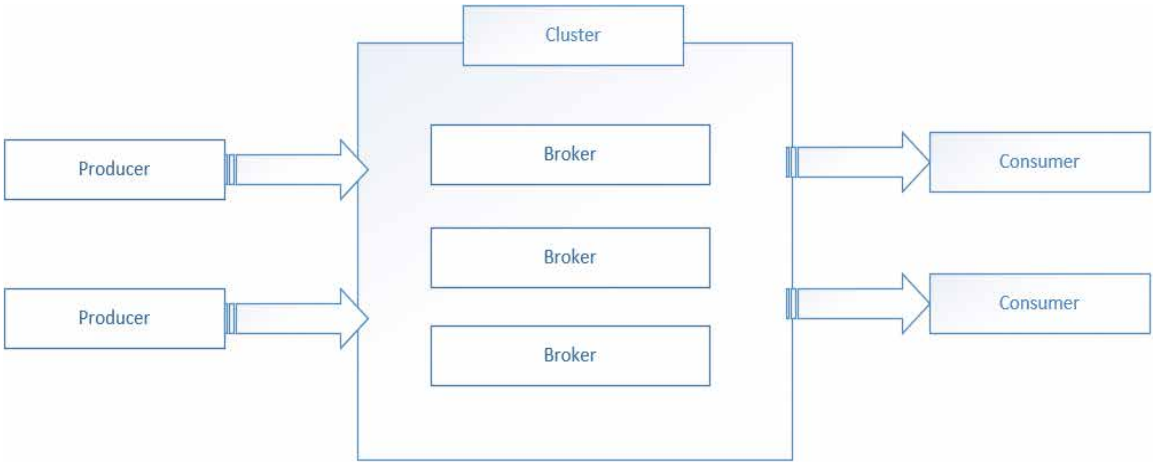


Figure 1: A high-level view of the components of the Kafka architecture

- **Kafka Broker**: A Kafka broker acts as a middleman between producers and consumers, hosting topics and partitions and enabling sending and receiving messages between them. The brokers in a typical production Kafka cluster can handle many reads/writes per second. Producers and consumers don't communicate directly. Instead, they communicate using these brokers. Thus, if one of the producers or consumers goes down, the communications pipeline continues to function as usual.

Figure 1 illustrates a high-level Kafka architecture. A Kafka cluster comprises one or more Kafka brokers. Although the Producers push messages into the Kafka topics in a Kafka broker, the Consumers pulls those messages off a Kafka topic.

When Not to Use Kafka

Despite being the most popular messaging platform and having several advantages, you should not use Kafka in any of the following use cases:

- Kafka's not a good choice if you need your messages processed in a particular order. To process messages in a specific order; you should have one consumer and one partition. Instead, in Kafka, you have multiple consumers and partitions and so it isn't an ideal choice in this use case.
- Kafka isn't a good choice if you only need to process a few messages per day (maybe up to several thousand). Instead, you can take advantage of traditional messaging queues like RabbitMQ.
- Kafka is an overkill for ETL jobs when real-time processing is required because it isn't easy to perform data transformations dynamically.
- Kafka is also not a good choice when you need a simple task queue. Instead, it would be best if you leveraged RabbitMQ here.

Kafka isn't a replacement for a database, and it should never be used for long-term storage. Because Kafka stores redundant copies of data, it might be a costly affair as well. When you need data to be persisted in a database for querying, insertion, and retrieval, you should use a relational database like Oracle, SQL Server, or a non-relational database like MongoDB.

Setting Up Apache Kafka

First off, download the Apache Kafka setup file from the location mentioned earlier. Now switch to the Downloads folder in your computer and install the downloaded files one by one. Kafka is available as a zip file, so you must extract the archive to the folder of your choice. Assuming you've already downloaded and installed 7-zip and Java in your computer, you can proceed with setting up and running Apache Kafka.

Now follow the steps outlined below:

1. Switch to the Kafka config directory in your computer. It is D:\kafka\config in my computer.
2. Open the file server.properties.
3. Find and replace the line "log.dirs=/tmp/kafka-logs" with "log.dirs=D:/kafka/kafka-logs", as shown in **Figure 2**.
4. Save and close the server.properties file.
5. Now open the file zookeeper.properties.
6. Find and replace the line "dataDir=/tmp/zookeeper" with "dataDir=D:/kafka/zookeeper-data", as shown in **Figure 3**.
7. Save and close the file

By default, Kafka runs on the default port 9092 in your computer and connects to ZooKeeper at the default port 2181.

Switch to your Kafka installation directory and start Zookeeper using the following command:

```
.\bin\windows\zookeeper-server-start.bat
     config\zookeeper.properties
```

Figure 4 is how it looks when Zookeeper is up and running in your system:

Launch another command window and write the following command in there to start Kafka:

```
.\bin\windows\
kafka-server-start.bat
config\server.properties
```

When Kafka is up and running in your system, it looks like **Figure 5**.

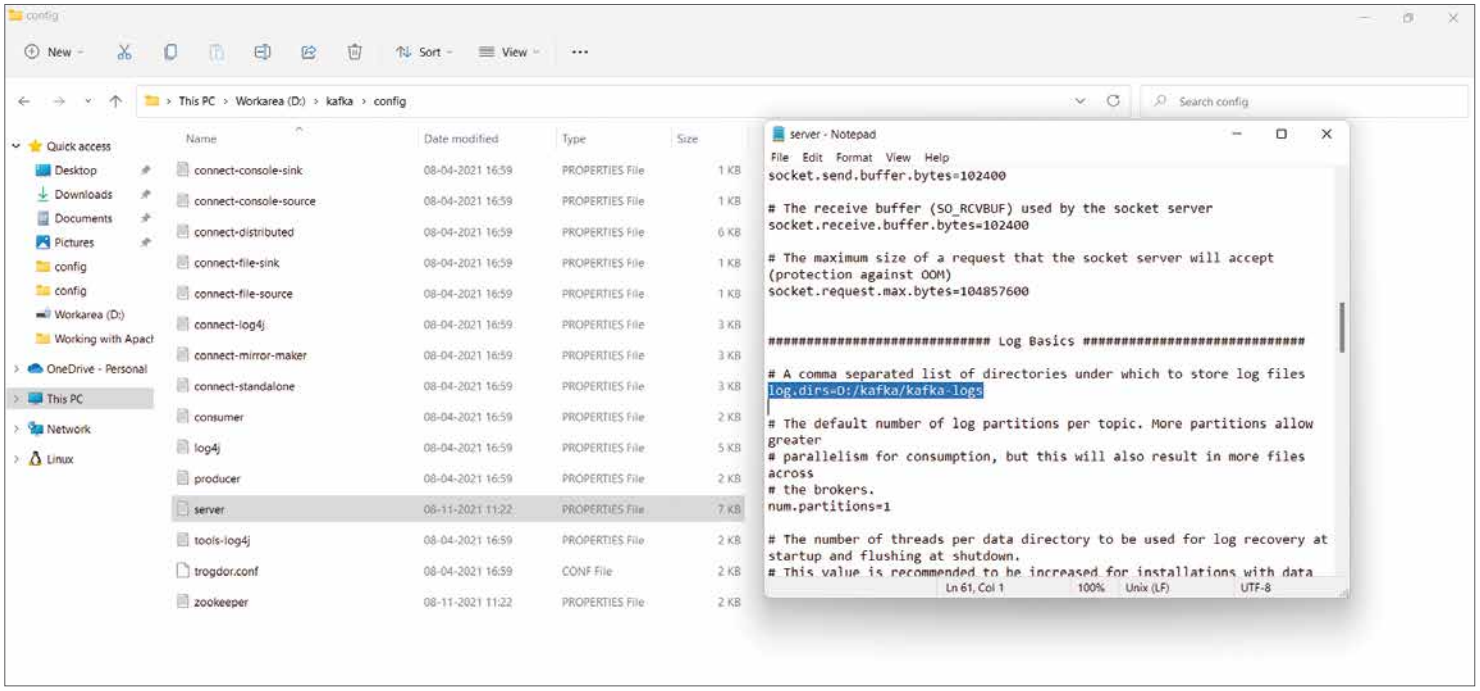


Figure 2: Setting up Apache Kafka. Specifying the logs directory

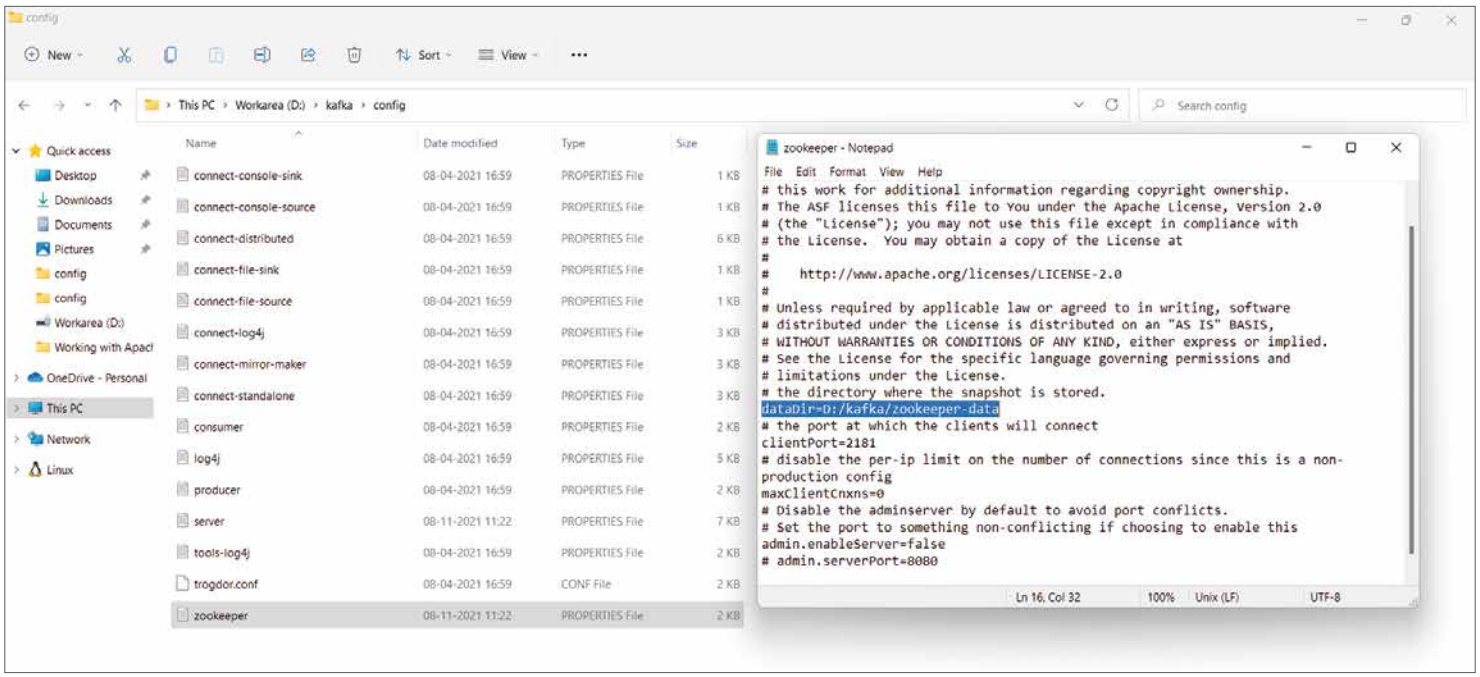


Figure 3: Setting up Apache Kafka: Specifying the data directory

Create Topic(s)

Now that Zookeeper and Kafka are both up and running, you should create one or more topics. To do this, follow the steps outlined below:

Launch a new command prompt window. Type the following command in there and press enter:

```
kafka-topics.bat --create --zookeeper
    localhost:2181 --replication-factor 1
    --partitions 1 --topic test
```

You can list all topics in a cluster using the following command:

```
.\bin\windows\kafka-topics.bat --list
    --zookeeper localhost:2181
```

Working with Apache Kafka in ASP.NET Core 6

In this section, you'll implement a simple Order Processing application. You'll build two applications: the producer ap-

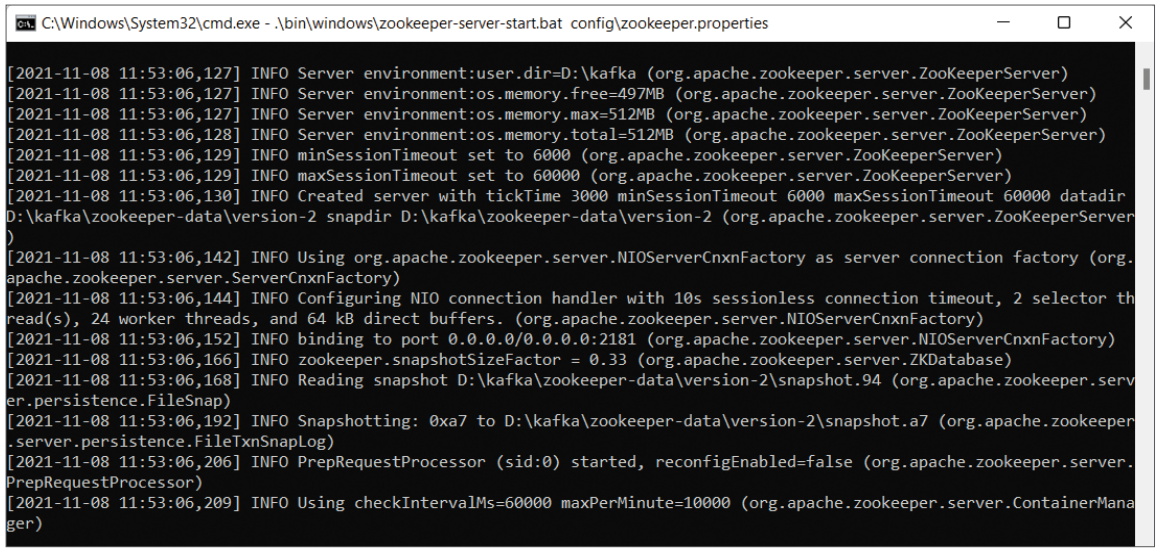


Figure 4: Zookeeper is up running at the default port 2181

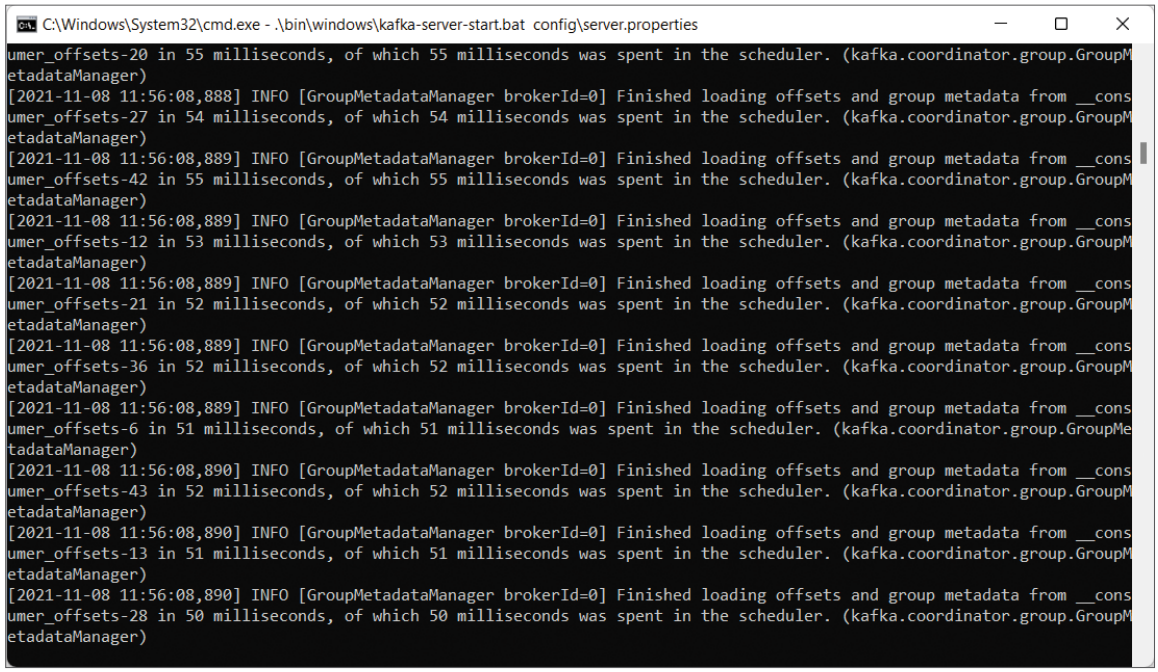


Figure 5: Kafka is up and running at the default port 9092.

plication and the consumer application. Both of these applications will be created using ASP.NET 6 in Visual Studio 2022 IDE.

Create a New ASP.NET 6 Project in Visual Studio 2022

Let's start building the producer application first. You can create a project in Visual Studio 2022 in several ways. When you launch Visual Studio 2022, you'll see the Start window. You can choose "Continue without code" to launch the main screen of the Visual Studio 2022 IDE.

To create a new ASP.NET 6 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 Preview IDE.
2. In the "Create a new project" window, select "ASP.NET Core Web API" and click Next to move on.
3. Specify the project name as ApacheKafkaProducerDemo and the path where it should be created in the "Configure your new project" window.
4. If you want the solution file and project to be created in the same directory, you can optionally check the "Place solution and project in the same directory" checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type as well. Ensure that the "Configure for HTTPS," "Enable Docker Support," and the "Enable OpenAPI support" checkboxes are unchecked because you won't use any of these in this example.
6. Click Create to complete the process.

Follow the same steps outlined above to create another ASP.NET Core 6 Web API project. Name this project ApacheKafka-

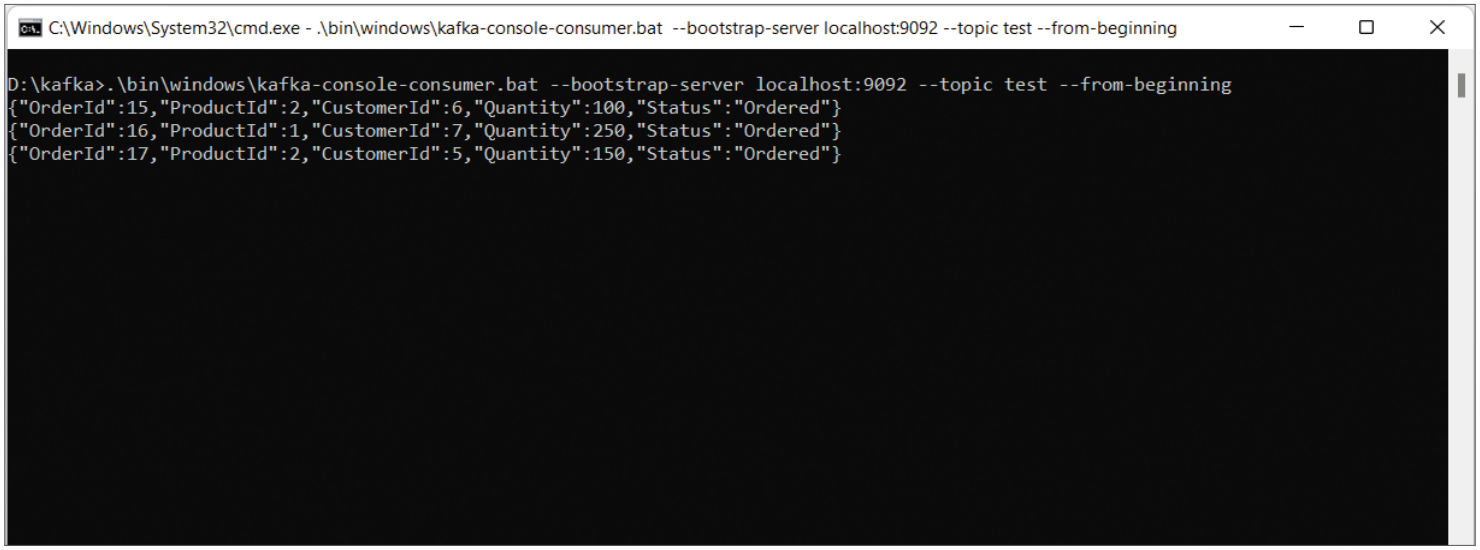


Figure 6: Installing Confluent.Kafka NuGet Package

ConsumerDemo. Note that you can also choose any meaningful name for both these projects.

You now have two ASP.NET Core 6 Web API projects: ApacheKafkaProducerDemo and the ApacheKafkaConsumerDemo.

Install NuGet Package(s)

So far so good. The next step is to install the necessary NuGet Package(s). To produce and consume messages, you need a client for Kafka. Use the most popular client: Confluent's Kafka .NET Client. To install the required packages into your project, right-click on the solution and select "Manage NuGet Packages for Solution...". Then type Confluent.Kafka in the search box, select the Confluent.Kafka package, and install it. You can see the appropriate screen in **Figure 7**.

Alternatively, you can execute the following command in the Package Manager Console:

```
PM> Install-Package Confluent.Kafka
```

Now you'll create the classes and interfaces for the two applications.

Building the ApacheKafkaProducerDemo Application

Create a class named OrderRequest in a file named OrderRequest.cs with the following code in there:

```
namespace ApacheKafkaProducerDemo
{
    public class OrderRequest
    {
        public int OrderId { get; set; }
        public int ProductId { get; set; }
        public int CustomerId { get; set; }
        public int Quantity { get; set; }
        public string Status { get; set; }
    }
}
```

Create a new controller named ProducerController in the ApacheKafkaProducerDemo application with the code found in **Listing 1** in it.

Listing 1: Create a new API Controller

```
using Confluent.Kafka;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Net;
using System.Text.Json;
using System.Threading.Tasks;
using System.Diagnostics;

namespace ApacheKafkaProducerDemo.Controllers {
  [Route("api/[controller]")]
  [ApiController]
  public class ProducerController: ControllerBase {
    private readonly string
    bootstrapServers = "localhost:9092";
    private readonly string topic = "test";

    [HttpPost]
    public async Task <IActionResult>
    Post([FromBody] OrderRequest orderRequest) {
      string message = JsonSerializer.Serialize
      (orderRequest);
      return Ok(await SendOrderRequest(topic, message));
    }
    private async Task < bool > SendOrderRequest
     (string topic, string message) {
      ProducerConfig config = new ProducerConfig {
        BootstrapServers = bootstrapServers,
          ClientId = Dns.GetHostName()
      };

      try {
        using(var producer = new ProducerBuilder
        <Null, string> (config).Build()) {
          var result = await producer.ProduceAsync
          (topic, new Message <Null, string> {
            Value = message
          });

          Debug.WriteLine($"Delivery Timestamp:
          {result.Timestamp.UtcDateTime}");
          return await Task.FromResult(true);
        }
      } catch (Exception ex) {
        Console.WriteLine($"Error occured: {ex.Message}");
      }

      return await Task.FromResult(false);
    }
  }
}
```

Building the ApacheKafkaConsumerDemo Application

Create a new class named OrderProcessingRequest in a file having the same name and a .cs extension with the following content in it:

```
namespace ApacheKafkaConsumerDemo
{
    public class OrderProcessingRequest
    {
        public int OrderId { get; set; }
        public int ProductId { get; set; }
        public int CustomerId { get; set; }
        public int Quantity { get; set; }
        public string Status { get; set; }
    }
}
```

Next,you'll create a hosted service to consume the messages. Create a class named ApacheKafkaConsumerService in another new file having the same name with a .cs extension, as found in **Listing 2**. This class should extend the IHostedService interface.

Register the Hosted Service

You should register the hosted service in the ConfigureServices method, as shown in the code snippet given below:

```
public void ConfigureServices
        (IServiceCollection services)
        {
            services.AddSingleton
            <IHostedService,
            ApacheKafkaConsumerService>();
```

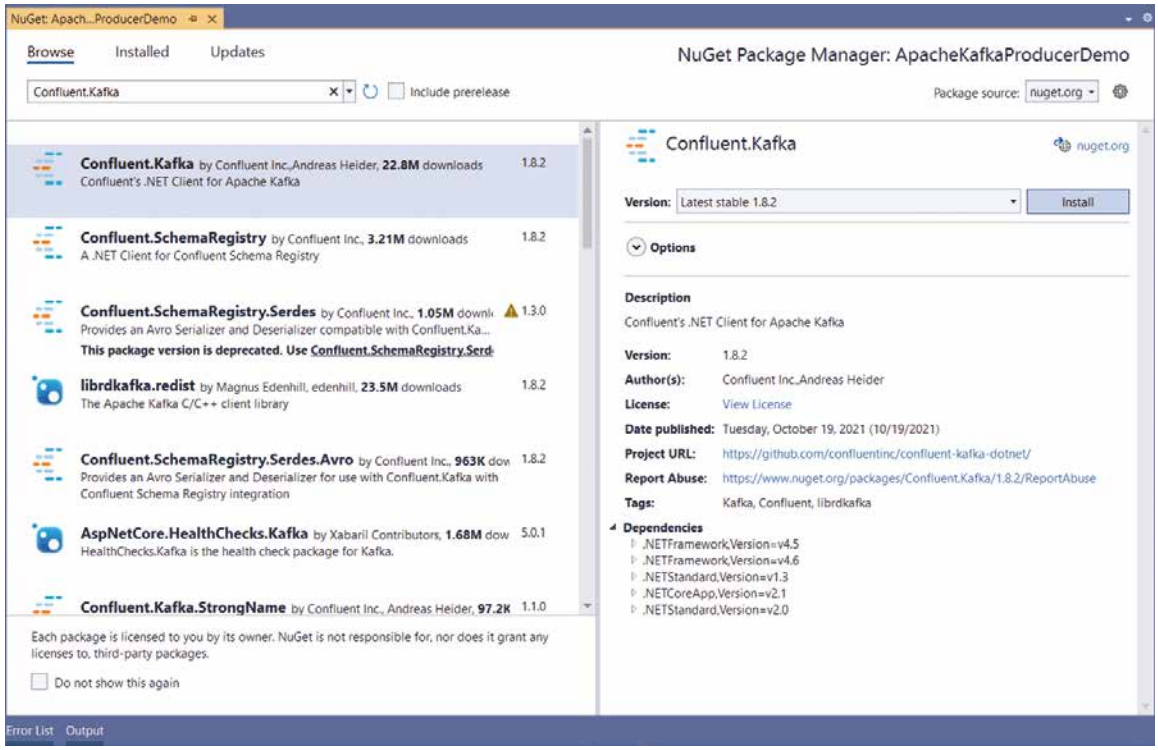


Figure 7: Send a POST request to the Producer application.

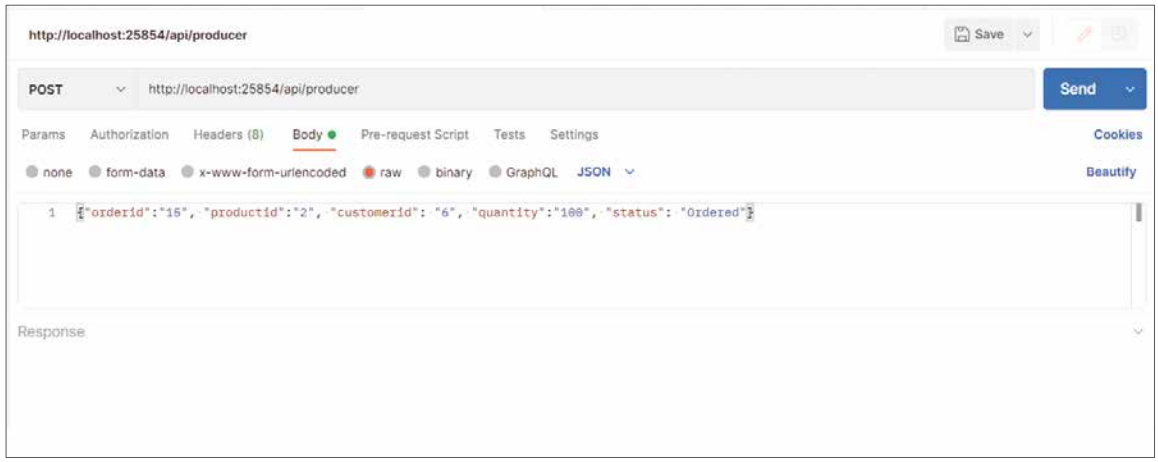


Figure 8: The breakpoint in the Producer application is hit

```
            services.AddControllers();
        }
```

Execute the Application

Set appropriate breakpoints in the source code of both applications so that you can debug them. You should run the producer and the consumer applications separately because they are part of different solutions. To run the application, follow these steps:

1. Execute the producer application.
2. Execute the consumer application.
3. Launch the Postman Http Debugger tool.
4. Send an HTTP POST request to the producer API using Postman, as shown in **Figure 8**.

Listing 2: Create an ApacheKafkaConsumer Service class

```
using Confluent.Kafka;
using Microsoft.Extensions.Hosting;
using System;
using System.Text.Json;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;

namespace ApacheKafkaConsumerDemo {
  public class ApacheKafkaConsumerService:
    IHostedService {
    private readonly string
    topic = "test";
    private readonly string
    groupId = "test_group";
    private readonly string
    bootstrapServers = "localhost:9092";

    public Task StartAsync
    (CancellationToken cancellationToken) {
      var config = new ConsumerConfig {
        GroupId = groupId,
          BootstrapServers = bootstrapServers,
          AutoOffsetReset = AutoOffsetReset.Earliest
      };

      try {
        using(var consumerBuilder =
       new ConsumerBuilder <Ignore, string>
        (config).Build()) {
          consumerBuilder.Subscribe(topic);
          var cancelToken =
          new CancellationTokenSource();

          try {
            while (true) {
              var consumer =
              consumerBuilder.Consume
              (cancelToken.Token);
              var orderRequest =
             JsonSerializer.Deserialize
             <OrderProcessingRequest>
             (consumer.Message.Value);
              Debug.WriteLine($"Processing Order Id:
             {orderRequest.OrderId}");
            }
          } catch (OperationCanceledException) {
            consumerBuilder.Close();
          }
        }
      } catch (Exception ex) {
        System.Diagnostics.Debug.WriteLine(ex.Message);
      }

      return Task.CompletedTask;
    }
    public Task StopAsync(CancellationToken
      cancellationToken) {
      return Task.CompletedTask;
    }
  }
}
```

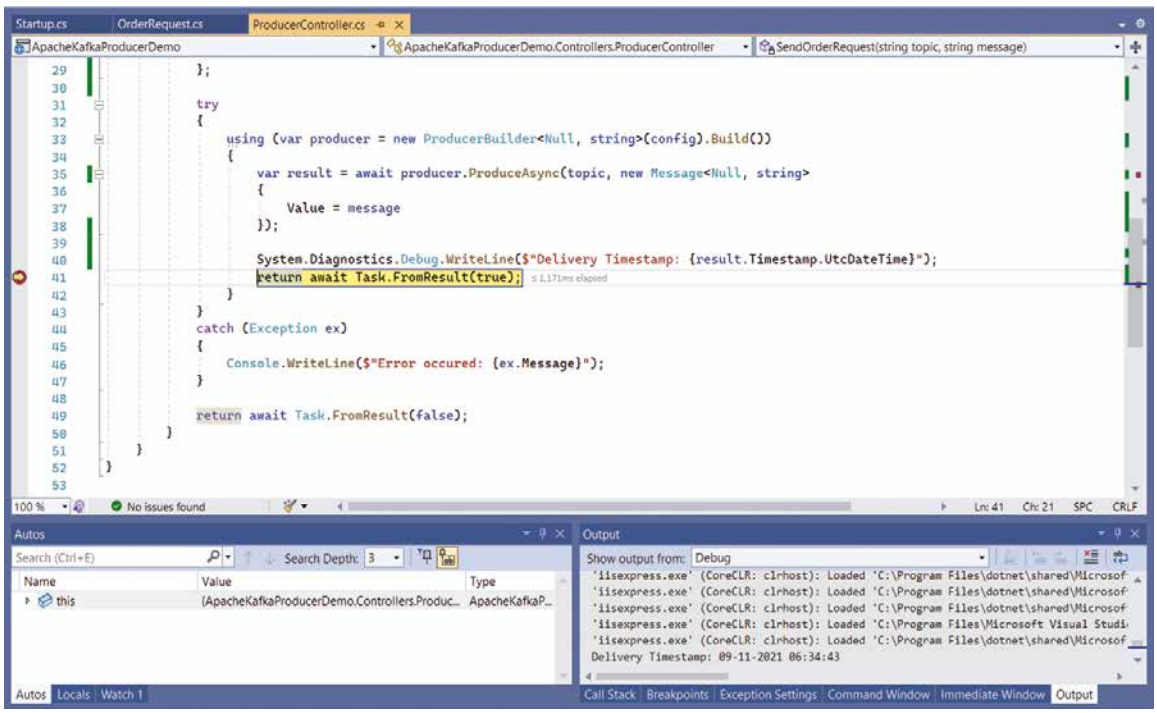


Figure 9: Displaying the Order ID of the order being processed in the Output window

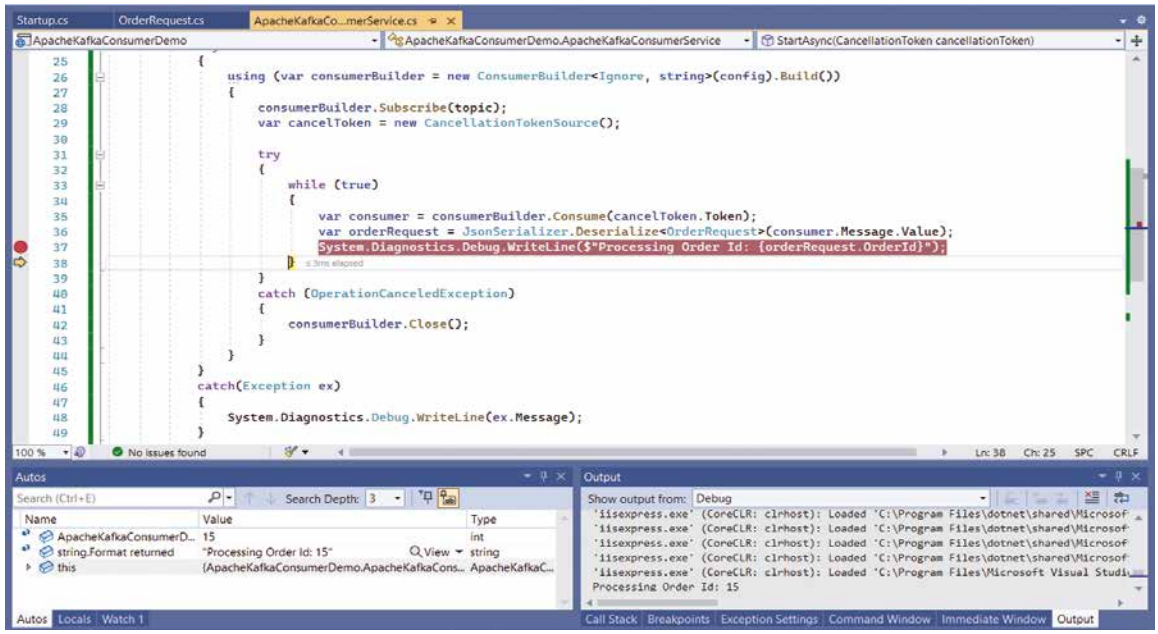


Figure 10: Displaying all messages in a topic

Figure 9 illustrates that the breakpoint has been hit in the producer application. Note the Timestamp value displayed in the Output window.

When you press F5, the breakpoint set in the consumer application will be hit and you can see the message displayed in the Output window, as shown in **Figure 10**.

Kafka CLI Administration

In this section we'll examine how we can perform a few administration tasks in Kafka.

Shut Down Zookeeper and Kafka

To shut down Zookeeper, use the zookeeper-server-stop.bat script, as shown below:

```
bin\windows\zookeeper-server-stop.bat
```

To shut down Kafka, you should use the kafka-server-stop.bat script as shown below:

```
bin\windows\kafka-server-stop.bat
```

> If you need a messaging system that's high-performant, resilient, and scalability, you'll be thrilled with Apache Kafka.

Display All Kafka Messages in a Topic

To display all messages in a particular topic, use the following command:

```
.\bin\windows\
kafka-console-consumer.bat
--bootstrap-server
localhost:9092 --topic
test --from-beginning
```

Where Should I Go from Here

Kafka is a natural choice if you're willing to build an application that needs high performant, resilient, and scalable messaging. This post walked you through building a simple Kafka producer and consumer using ASP.NET 6. Note that you can set up Apache Kafka using Docker as well. You can know more about Apache Kafka from the Apache Kafka Documentation.

Joydip Kanjilal

The Secrets of Manipulating CSV Files

It's 2022 and one of the most common file types I deal with on nearly a daily basis is the CSV file. If you told me this just a few short years ago, I would have told you: "The 80s called and they want their file format back." And yet here we are in the second decade of the 21st century and I DO deal with CSV more frequently than I would have ever expected. First, let's discuss just what CSV files are. Comma Separated Files (CSV) are text files that contain multiple records (rows), each with one or more elements (columns) separated by a comma character. Actually, the elements can be separated by any type of delimiter, not only a comma. For instance, another common file format is the Tab Separated Value (TSV) file where every element is separated by a tab character.

> The 80s called and they want their file format back.

So why is there this sudden demand for skills when dealing with CSV files? As The Dude would say: It's the science, man. And by that, I mean the Data Science. In our current state of development, we deal with huge quantities of data, and often this data is shared between organizations. CSV files present a unique set of opportunities for sharing large quantities of data as they're dense and contain little of the wasted content that's commonly found in JSON or XML files. They also compress rather nicely, which lowers bandwidth uses. **Figure 1** shows an example of a simple CSV file containing movie data.

By the end of this article, you'll be intimately familiar with this data. You'll learn how to read, write, and format Visual Studio files based on this data.

Movie Data Sample

As stated above, this article will be all about reading and writing movie data formatted in various CSV formats. The following class code represents the data:

```
public class Movie
{
 public string Name { get; set; } = "";
 public string Director { get; set; } = "";
 public DateTime DateReleased { get; set; }
 public decimal
  BoxOfficeGross { get; set; } = 0.0m;
}

public static List<Movie> GetMovies()
{
 var movies = new List< Movie >();

 movies.Add(new Movie (){Name =
  "American Graffiti",
     Director = "George Lucas",
  DateReleased = new DateTime(1977,5,23),
  BoxOfficeGross = 123456});

  movies.Add(new Movie () { Name = "Star Wars",
    Director = "George Lucas",
    DateReleased = new DateTime(1977, 5, 23),
    BoxOfficeGross = 123456 });

  movies.Add(new Movie () { Name =
   "Empire Strikes Back",
    Director = "Irving Kirshner",
    DateReleased = new DateTime(1977, 5, 23),
    BoxOfficeGross = 123456 });

  movies.Add(new Movie (){ Name =
    "Return of the Jedi",
     Director = "Richard Marquand",
     DateReleased = new DateTime(1977, 5, 23),
     BoxOfficeGross = 123456
     });
     return movies;
}
```

Introducing CSVHelper

A few years ago, my team began building a Data Analytics platform for our Data Scientists to use. The data was hosted in a platform called Snowflake that uses CSV files as a mechanism for loading data into their cloud services. When this need arose, I did what all good developers do: I searched for a tool that would help me deal with CSV files.

This is where I came across a .NET library called CSVHelper. This open-source tool, written by developer Josh Close (any many others), is simple to use yet powerful enough to deal with many data types of CSV scenarios that have presented themselves over the years.

Getting up and running with CSVHelper is simple. The following steps demonstrate how to bootstrap a .NET application capable of manipulating CSV files.

Bootstrapping CSVHelper

There are only two step to bootstrapping CSVHelper:

1. Create a new Console Application
2. Install CSVHelper via the NuGet Package Manager Console using the following command:

```
Install-Package CsvHelper
```

Now you're ready to begin manipulating CSV files.

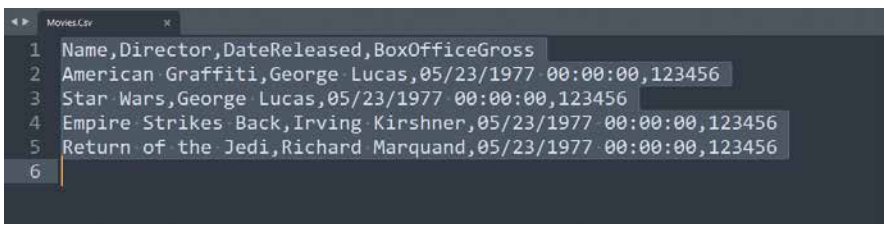


Figure 1: Sample movie data in CSV format

Rod Paddock
rodpaddock@dashpoint.com

Rod Paddock founded Dash Point Software, Inc. in 2001 to develop high-quality custom software solutions. With 30+ years of experience, Rod's current and past clients include: Six Flags, First Premier Bank, Microsoft, Calamos Investments, The US Coast Guard, and US Navy. Along with developing software, Rod is a well-known author and conference speaker. Since 1995, Rod has given talks, training sessions, and keynotes in the US, Canada, and Europe. Rod has been Editor-in-Chief of CODE Magazine since 2001.

Writing CSV Files

Once you've created your basic project, you can start by outputting a collection of data to a CSV file. The following code demonstrates the simplest mechanism for writing a collection of movie records to a CSV file.

```
public static void WriteCsvFile(
List<Movie> dataToWrite, string outputFile)
{
  var config =
    new CsvConfiguration(
  CultureInfo.InvariantCulture);

  using (var writer =
    new StreamWriter(outputFile))
  using (var csv =
     new CsvWriter(writer, config))
    {
     csv.WriteRecords(dataToWrite);
    }
}
```

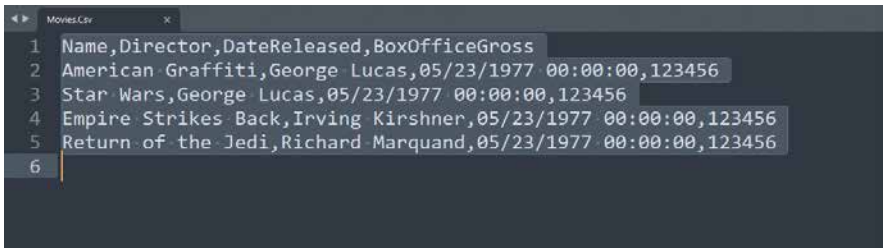


Figure 2: Movie Data Output as CSV file.

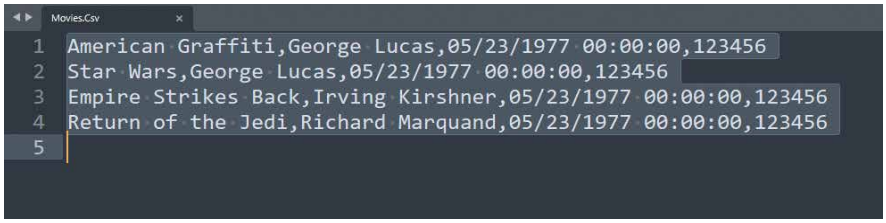


Figure 3: A CSV file with no header

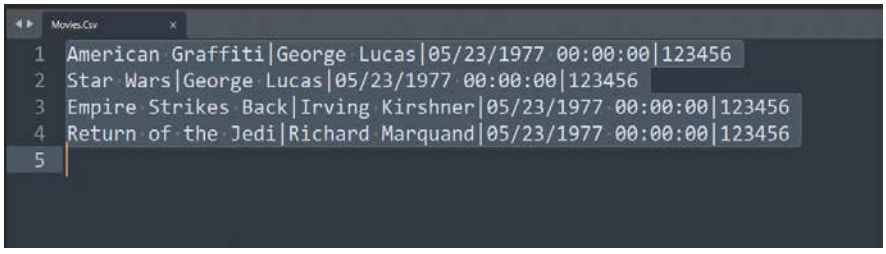


Figure 4: The CSV file with PIPE delimiter

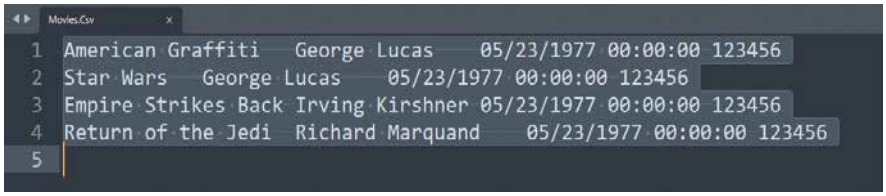


Figure 5: The CSV file with TAB delimiter

When you examine this code, take notice of the following items:

- The code creates a CSVConfiguration object. This object will be used to control the output of your CSV file.
- The file opens a StreamWriter that controls where your file will be written.
- The code then creates a CSVWriter object passing in the configuration object. This Writer sends your data to the stream opened by the writer using the passed-in configuration settings.
- Finally, the call to WriteRecords routine takes an IEnumarable collection and writes to the CSV fille.

The output of this set of code can be found in **Figure 2**.

Configuring Writer Options

As stated earlier, the CSVWriter accepts a configuration object that's used to control output options. A few of the key options will be covered next.

Header Column

You may or may not want to include a header file in your CSV files. By default, CSVHelper adds the name of your class' properties in a header row. You can turn off the header by setting it with the following code:

```
config.HasHeaderRecord = false;
```

Figure 3 shows the results of this option.

Changing Delimiters

One of the more common options is the delimiter used between each data element. By default, CSVHelper delimits data comma characters. The following three examples show how you can change the delimiter to the PIPE, TAB, and a "crazy" delimiter.

- Changing the delimiter to PIPE:

```
config.Delimiter = "|";
```

Figure 4 shows the PIPE delimiter in action.

- Changing the delimiter to TAB:

```
config.Delimiter = "\t";
```

Figure 5 shows the TAB delimiter in action.

- Creating a "Crazy" delimiter (This is just to demonstrate that your delimiter can be anything you desire):

```
config.Delimiter = "[[YES_IM_A_DELIMETER]]";
```

Figure 6 shows the "Crazy" delimiter doing its thing.

Quote Delimiting

I've found in many situations that my data needs to have each data element wrapped in quotation marks. This is especially true when your data contains delimiters within their fields, e.g., commas. CSVHelper allows you to quote-delimit your data using the following options.

```
config.ShouldQuote = args => true;
```

Figure 7 shows the CSV with quoted content.

Formatting Output with Map Classes

Another very handy tool is the ability to control the output sent to your file. By default, CSVHelper outputs elements by reflecting on the class they come from and creating columns for each property. There are many situations where you may want to export a limited set of properties or you wish to change the order of the output files.

This is where mapping classes come in. When exporting data CSVHelper can accept a mapping object derived from the ClassMap class. The following code demonstrates a ClassMap that limits the data exported to two properties.

```
public class MovieOutputClassMap
   : ClassMap<Movie>
{
  public MovieOutputClassMap()
  {
    Map(m => m.Name);
    Map(m => m.DateReleased);

   }
}
```

Once you've built your class map, you need to apply it to your writer. This is done using two commands. The first one creates an instance of your class map.

```
var classMap = new MovieOutputClassMap();
```

The second registers it with the writer Context property:

```
csv.Context.RegisterClassMap(classMap);
```

The full writer code is shown below:

```
public static void
  WriteCsvFile(List<Movie> dataToWrite,
   string outputFile)
{
var config =
   new CsvConfiguration(CultureInfo.InvariantCulture);

  //include header
  config.HasHeaderRecord = false;

  //change delimiter
  config.Delimiter = "|";

  //quote delimit
  config.ShouldQuote = args => true;


  //changing the order of fields
  var classMap = new MovieOutputClassMap();

  using (var writer =
                new StreamWriter(outputFile))

  using (var csv =
    new CsvWriter(writer, config))
     {
      csv.Context.RegisterClassMap(classMap);
      csv.WriteRecords(dataToWrite);
     }
}
```

Figure 8 shows the CSV file with two columns.

You can also use a class map to reorder your output

```
public class MovieOutputClassMap
   : ClassMap<Movie>
  {
      public MovieOutputClassMap()
      {
          Map(m => m.Name);
          Map(m => m.DateReleased);
          Map(m => m.Director);
          Map(m => m.BoxOfficeGross);
       }
  }
```

Figure 9 shows the CSV file with its columns reordered.

Along with altering the number of columns exported and changing the ordinal position of them, you can also control the text that's emitted into the CSV stream. Altering the output (and input) is done using a class that implements the **ITypeConverter** interface.

The code below demonstrates creating a type converter that alters the output of the DateReleased property removing the time component.

This code receives the property's value and returns a string using the ConvertToString aspect of the type converter. There's also a corollary for reading these values from strings via an implementation of the ConvertFromString function.

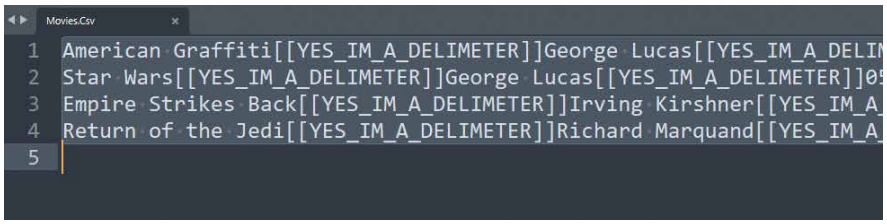


Figure 6: The CSV file with the CRAZY delimiter

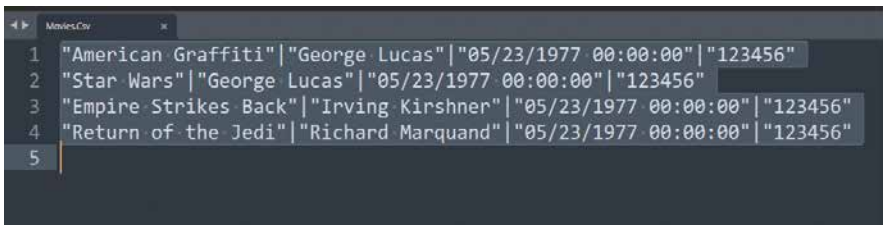


Figure 7: The CSV file with quoted content

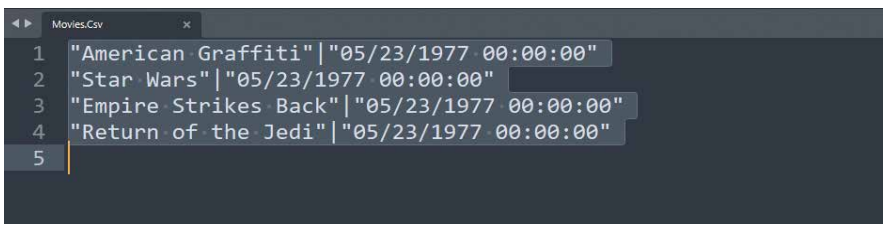


Figure 8: CSV file with only two columns exported

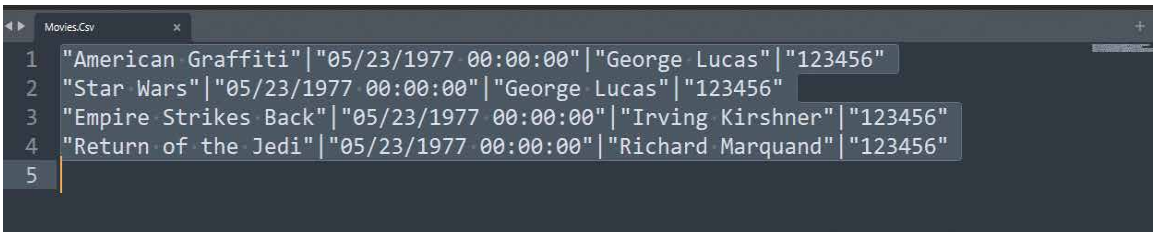


Figure 9: The CSV file columns reordered

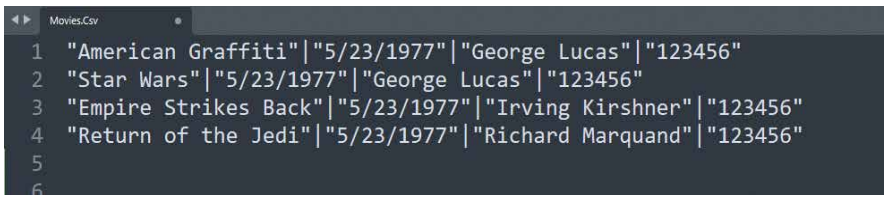


Figure 10: The CSV file with the date formatting altered

```
public class DateOutputConverter : ITypeConverter
{
  public object ConvertFromString(string text,
     IReaderRow row, MemberMapData memberMapData)
  {
    throw new NotImplementedException();
   }
   public string ConvertToString(
     object value,
     IWriterRow row,
     MemberMapData memberMapData)
    {
      var retval =
        ((DateTime) value).ToString("d");
      return retval;
     }
  }
```

Once you've created your converter, you attach it to your column via the mapping class. The following code shows how to attach a converter to a property map.

```
public class MovieOutputClassMap :
    ClassMap<Movie>
{
  public MovieOutputClassMap()
  {
    Map(m => m.Name);
    Map(m => m.DateReleased).TypeConverter(new
       DateOutputConverter());
    Map(m => m.Director);
    Map(m => m.BoxOfficeGross);
  }
```

Figure 10 shows the CSV file with the date formatting altered.

Reading CSV Files

Now that you have a basic understanding of writing CSV files, you can turn your sights to reading CSV files. There are two primary mechanisms for reading a file. The first is to open the file and iterate through it one record at a time.

When you examine this set of code for reading files, take notice of the following items:

- The code creates a CSVConfiguration object. This object is used to control how the reader manipulated your CSV data as it was read.
- The file opens a StreamReader, which controls where your file will be read from.
- The code then creates a CSVReader object passing in the configuration object. This reader is used to iterate through your CSV file one record at a time.
- The code iterates the file using the Read() function, which moves down the file one record at a time. Note that the code does a Read() immediately, to skip the record header.
- Finally, the code uses various Getter functions to read data from each column.

```
public static List<Movie>
    ManualReadCsvFile(string inputFile)
{

 var retval = new List<Movie>();
 var config = new CsvConfiguration(
   CultureInfo.InvariantCulture);


 using (var reader =
          new StreamReader(inputFile))

 using (var csv =
  new CsvReader(reader, config))
  {
   //skip the header
   csv.Read();
   while (csv.Read())
   {
     var movie = new Movie();
     movie.Name = csv.GetField(0);
     movie.Director = csv.GetField(1);
     movie.DateReleased
 =csv.GetField<DateTime>(2);
       movie.BoxOfficeGross
 =csv.GetField<decimal>(3);
       retval.Add(movie);
    }
   }
  return retval;
}
```

Another and much simpler way to read a file is to use CSVHelper's built-in mechanism for iterating through a file automatically transforming CSV records into to .NET classes.

When you examine this set of code for reading files, take notice of the following items:

- The code creates a CSVConfiguration object. This object is used to control how the reader manipulated your CSV data as it was read.
- The file opens a StreamReader, which controls where your file will be read from.
- The code then creates a CSVReader object passing in the configuration object. This reader is used to iterate through your CSV file one record at a time.
- The code then reads all the records using the GetRecords<T> method. This function returns an IEnumerable collection.
- The collection is then added to the functions return value via the AddRange() method.

```
public static List<Movie> ReadCsvFile(
  string inputFile)
{
var retval = new List<Movie>();
var config =
  new CsvConfiguration
     (CultureInfo.InvariantCulture);

  using (var reader =new StreamReader(inputFile))
  using (var csv =
   new CsvReader(reader, config))
     {
      retval.AddRange(csv.GetRecords<Movie>());
     }
  return retval;
 }
```

As you can see, this style of code is much simpler to deal with.

> Simple code is always better, both while you're writing it and when you come back to it later.

You can also use class maps to change the order of how CSV elements are read from your CSV file and are applied to the returned object's properties. The following class map reads content from the CSV created earlier in this article. Notice the column order.

```
public class MovieInputClassMap : ClassMap<Movie>
    {
        public MovieInputClassMap()
        {
            Map(m => m.Name);
            Map(m => m.DateReleased);
            Map(m => m.Director);
            Map(m => m.BoxOfficeGross);
        }
    }
```

The code used to attach a class map is exactly like the writer. You simply create an instance of the class map and apply it to the CSVReader's Context property:

```
public static List<Movie> ReadCsvFile(
  string inputFile)
{
  var retval = new List<Movie>();
  var config =
    new CsvConfiguration(CultureInfo.InvariantCulture);

    var classMap = new MovieInputClassMap();

    using (var reader =
        new StreamReader(inputFile))

    using (var csv =
      new CsvReader(reader, config))
       {
      csv.Context.RegisterClassMap(classMap);
                retval.AddRange(csv.GetRecords<Movie>());
           }

           return retval;
       }
```

Conclusion

As you can see, using CSVHelper greatly simplifies the process of reading and writing CSV files. This library has a good balance of simple-to-use yet very capable tools. I highly recommend exploring more capabilities of this object.

So now you may be asking yourself how you can exploit these tools and techniques in the Data Science space. Well, that's where the next article comes in. In a future article, I'll demonstrate streaming this data into Snowflake via an S3 bucket and how to bulk-load data into Postgres using the same tools. Thanks for exploring the non-glamorous world of CSV files with me.

Rod Paddock

Minimal APIs in .NET 6

Building REST APIs has become central to many development projects. The choice for building those projects is wide, but if you're a C# developer, the options are more limited. Controller-based APIs have been the most common for a long time, but .NET 6 changes that with a new option. Let's talk about it.

Shawn Wildermuth
shawn@wildermuth.com
wildermuth.com
twitter.com/shawnwildermut

Shawn Wildermuth has been tinkering with computers and software since he got a Vic-20 back in the early '80s. As a Microsoft MVP since 2003, he's also involved with Microsoft as an ASP.NET Insider and ClientDev Insider. He's the author of over twenty Pluralsight courses, written eight books, an international conference speaker, and one of the Wilder Minds. You can reach him at his blog at http://wildermuth.com. He's also making his first, feature-length documentary about software developers today called "Hello World: The Film." You can see more about it at http://helloworldfilm.com.

How'd We Get Here?

Connecting computers has been a problem since the first steps of distributed computing some fifty years ago (see **Figure 1**). Yeah, that makes me feel old too. Remote Procedure Calls were as important as APIs in modern development. With REST, OData, GraphQL, GRPC, and the like, we have a lot of options to create ways of communicating between apps.

Although lots of these technologies are thriving, using REST as a way of communicating is still a stalwart in today's development world. Microsoft has had a number of solutions for creating REST APIs over the years, but for the past decade or so, Web API has been the primary tool. Based on the ASP.NET MVC framework, Web API was meant to treat REST verbs and nouns as first-class citizens. Being able to create a class that represents a surface area (often tied to a "noun" in the REST sense) that's tied together with a routing library is still a viable way to build APIs in today's world.

One of the drawbacks to the Web API framework (e.g., Controllers), is that there's a bit of ceremony involved for small APIs (or microservices). For example, Controllers are classes that represent one or more possible calls to an API:

```
[Route("api/[Controller]")]
[Authorize(AuthenticationSchemes =
  JwtBearerDefaults.AuthenticationScheme)]
public class OrdersController : Controller
{
  readonly IDutchRepository _repository;
  readonly ILogger<OrdersController> _logger;
  readonly IMapper _mapper;
  readonly UserManager<StoreUser> _userManager;

  public OrdersController(
    IDutchRepository repository,
    ILogger<OrdersController> logger,
    IMapper mapper,
    UserManager<StoreUser> userManager)
  {
    _repository = repository;
    _logger = logger;
    _mapper = mapper;
    _userManager = userManager;
  }

  [HttpGet]
  public IActionResult Get(
    bool includeItems = true)
  {
    try
    {
      var username = User.Identity.Name;

      var results = _repository
        .GetOrdersByUser(username,
                         includeItems);

      return Ok(_mapper
        .Map<IEnumerable<OrderViewModel>>(
          results));
    }
    catch (Exception ex)
    {
      _logger.LogError($"Failed : {ex}" );
      return BadRequest($"Failed");
    }
  }
  ...
```

This code is typical of Web API Controllers. But does that mean it's bad? No. For larger APIs and ones that have advanced needs (e.g., rich authentication, authorization, and versioning), this structure works great. But for some projects, a simpler way to build APIs is really needed. Some of this pressure is coming from other frameworks where building APIs feels smaller, but it's also driven by wanting to be able to design/prototype APIs more quickly.

This need isn't all that new. In fact, the Nancy framework (https://github.com/NancyFx/Nancy) was a C# solution for mapping APIs calls way back then (although it's deprecated now). Even newer libraries like Carter (https://github.com/CarterCommunity/Carter) are trying to accomplish the same thing. Having efficient and simple ways to create APIs is a necessary technique. You shouldn't take Minimal APIs as the "right" or "wrong" way to build APIs. Instead, you should see it as another tool to build your APIs.

Enough talk, let's dig into how it works.

What Are Minimal APIs?

The core idea behind Minimal APIs is to remove some of the ceremony of creating simple APIs. It means defining lambda expressions for individual API calls. For example, this is as simple as it gets:

```
app.MapGet("/", () => "Hello World!");
```

This call specifies a route (e.g., "/") and a callback to execute once a request that matches the route and verb are matched. The method **MapGet** is specifically to map a HTTP GET to the callback function. Much of the magic is in the type inference that's happening. When we return a string (like in this example), it's wrapping that in a 200 (e.g., OK) return result.

How do you even call this? Effectively, these mapping methods are exposed. They're extension methods on the **IEndpointRouteBuilder** interface. This interface is exposed by the **WebApplication** class that's used to create a new Web server application in .NET 6. But I can't really dig into this without first talking about how the new Startup experience in .NET 6 works.

The New Startup Experience

A lot has been written about the desire to take the boilerplate out of the startup experience in C# in general. To this end, Microsoft has added something called "Top Level Statements" to C# 10. This means that the **program.cs** that you rely on to start your Web applications don't need a **void Main()** to bootstrap the app. It's all implied. Before C# 10, a startup looked something like this:

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace Juris.Api
{
  public class Program
  {
    public static void Main(string[] args)
    {
      CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder
      CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
         .ConfigureWebHostDefaults(webBuilder =>
         {
           webBuilder.UseStartup<Startup>();
         });
  }
}
```

The need for a class and a **void Main** method that bootstraps the host to start the server is how we've been writing ASP.NET in the .NET Core way for a few years now. With top-level statements, they want to streamline this boilerplate, as seen below:

```
var builder = WebApplication.CreateBuilder(args);

// Setup Services

var app = builder.Build();

// Add Middleware

// Start the Server
app.Run();
```

Instead of a Startup class with places to set up services and middleware, it's all done in this very simple top-level program. What does this have to do with Minimal APIs? The **app** that the **builder** object builds supports the **IEndpointRouteBuilder** interface. So, in our case, the set up to the APIs is just the middleware:

```
var builder = WebApplication.CreateBuilder(args);

// Setup Services

var app = builder.Build();

// Map APIs
app.MapGet("/", () => "Hello World!");

// Start the Server
app.Run();
```

Let's talk about the individual features here.

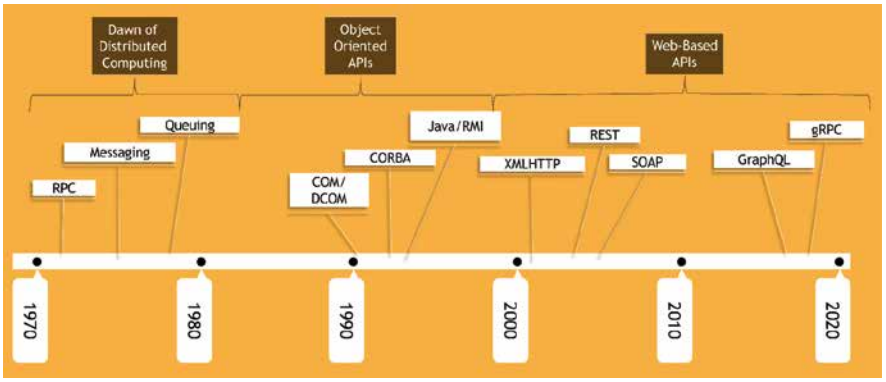


Figure 1: History of APIs

Routing

The first thing you might notice is that the pattern for mapping API calls looks a lot like MVC Controllers' pattern matching. This means that Minimal APIs look a lot like controller methods. For example:

```
app.MapGet("/api/clients", () => new Client()
{
  Id = 1,
  Name = "Client 1"
});

app.MapGet("/api/clients/{id:int}",
  (int id) => new Client()
{
  Id = id,
  Name = "Client " + id
});
```

Simple paths like the "/api/clients" point at simple URI paths, whereas using the parameter syntax (even with constraints) continues to work. Notice that the callback can accept the ID that's mapped from the URI just like MVC Controllers. One thing to notice in the lambda expression is that the parameter types are inferred (like most of C#). This means that because you're using a URL parameter (e.g., **id**), you need to type the first parameter. If you didn't type it, it would try to guess the type in the lambda expression:

```
app.MapGet("/api/clients/{id:int}",
  (id) => new Client()
{
  Id = id, // Doesn't Work
  Name = "Client " + id
});
```

This doesn't work because without the hint of type, the first parameter of the lambda expression is assumed to be an instance of "**HttpContext**". That's because, at its lowest level, you can manage your own response to any request with the context object. But for most of you, you'll use the parameters of the lambda expression to get help in mapping objects and parameters.

Using Services

So far, the APIs calls you've seen aren't anything like real world. In most of those cases, you want to be able to use common services to execute calls. This brings me to how to use Services in Minimal APIs. You may have noticed earlier that I'd left a space to register services before I built the **WebApplication**:

```
var bldr = WebApplication.CreateBuilder(args);

// Register Services Here

var app = bldr.Build();
```

You can just use the builder object to access the services, like so:

```
var bldr = WebApplication.CreateBuilder(args);

// Register Services
bldr.Services.AddDbContext<JurisContext>();
bldr.Services.AddTransient<IJurisRepository,
                           JurisRepository>();

var app = bldr.Build();
```

Here you can see that you can use the Services object on the application builder to add any services you need (in this case, I'm adding an Entity Framework Core context object and a repository that I'll use to execute queries. To use these services, you can simply add them to the lambda expression parameters to use them:

```
app.MapGet("/clients",
  async (IJurisRepository repo) => {
    return await repo.GetClientsAsync();
  });
```

By adding the required type, it will be injected into the lambda expression when it executes. This is unlike Controller-based APIs in that dependencies are usually defined at the class level. These injected services don't change how services are handled by the service layer (i.e., Minimal APIs still create a scope for scoped services). When you're using URI parameters, you can just add the services required to the other parameters. For example:

```
app.MapGet("/clients/{id:int}",
  async (int id, IJurisRepository repo) => {
    return await repo.GetClientAsync(id);
  });
```

This requires you think about the services you require for each API call separately. But it also provides the flexibility to use services at the API level.

Verbs

So far, all I've looked at are HTTP GET APIs. There are methods for the different types of verbs. These include:

- MapPost
- MapPut
- MapDelete

These methods work identically to the **MapGet** method. For example, take this call to POST a new client:

```
app.MapPost("/clients",
  async (Client model,
         IJurisRepository repo) =>
  {
    // ...
  });
```

Notice that the model in this case doesn't need to use attributes to specify **FromBody.** It infers the type if the shape matches the type requested. You can mix and match all of what you might need (as seen in **MapPut**):

```
app.MapPut("/clients/{id}",
  async (int id,
         ClientModel model,
         IJurisRepository repo) =>
  {
    // ...
  });
```

For other verbs, you need to handle mapping of other verbs using MapMethods:

```
app.MapMethods("/clients", new [] { "PATCH" },
  async (IJurisRepository repo) => {
    return await repo.GetClientsAsync();
  });
```

Notice that the **MapMethods** method takes a path, but also takes a list of verbs to accept. In this case, I'm executing this lambda expression when a PATCH verb is received. Although you're creating APIs separately, most of the same code that you're familiar with will continue to work. The only real change is how the plumbing finds your code.

Using HTTP Status Codes

In these examples, so far, you haven't seen how to handle different results of an API action. In most of the APIs I write, I can't assume that it succeeds, and throwing exceptions isn't the way that I want to handle failure. To that end, you need a way of controlling what status codes to return. These are handled with the **Results** static class. You simply wrap your result with the call to Results and the status code:

```
app.MapGet("/clients",
  async (IJurisRepository repo) => {
    return Results.Ok(
      await repo.GetClientsAsync());
  });
```

Results supports most status codes you'll need, like:

- **Results.Ok**: 200
- **Results.Created**: 201
- **Results.BadRequest**: 400
- **Results.Unauthorized**: 401
- **Results.Forbid**: 403
- **Results.NotFound**: 404
- Etc.

In a typical scenario, you might use several of these:

```
app.MapGet("/clients/{id:int}",
  async (int id, IJurisRepository repo) => {

    try {
      var client = await repo.GetClientAsync(id);
      if (client == null)
      {
        return Results.NotFound();
      }
      return Results.Ok(client);
    }
    catch (Exception ex)
    {
      return Results.BadRequest("Failed");
    }
  });
```

If you're going to pass in a delegate to the **MapXXX** classes, you can simply have them return an **IResult** to require a status code:

```
app.MapGet("/clients/{id:int}", HandleGet);

async Task<IResult> HandleGet(int id,
  IJurisRepository repo)
{
  try
  {
    var client = await repo.GetClientAsync(id);
    if (client == null) return Results.NotFound();
    return Results.Ok(client);
  }
  catch (Exception)
  {
    return Results.BadRequest("Failed");
  }
}
```

Notice that because you're **async** in this example, you need to wrap the **IResult** with a **Task** object. The resulting return is an instance of **IResult**. Although Minimal APIs are meant to be small and simple, you'll quickly see that, pragmatically, APIs are less about how they're instantiated and more about the logic inside of them. Both Minimal APIs and Controller-based APIs work essentially the same way. The plumbing is all that changes.

Securing Minimal APIs

Although Minimal APIs work with Authentication and Authorization middleware, you may still need a way to specifying, on an API-level, how security should work. If you're coming from Controller-based APIs, you might use the **Authorize** attribute to specify how to secure your APIs, but without controllers, you're left to specify them at the API level. You do this by calling methods on the generated API calls. For example, to require authorization:

```
app.MapPost("/clients",
  async (ClientModel model,
         IJurisRepository repo) =>
  {
    // ...
  }).RequireAuthorization();
```

This call to **RequireAuthorization** is tantamount to using the Authorize filter in Controllers (e.g., you can specify which Authentication scheme or other properties you need). Let's say you're going to require authentication for all calls:

```
bldr.Services.AddAuthorization(cfg => {
  cfg.FallbackPolicy =
    new AuthorizationPolicyBuilder()
      .RequireAuthenticatedUser()
      .Build();
});
```

You'd then not need to add **RequireAuthentication** on every API, but you could override this default by allowing anonymous for other calls:

```
app.MapGet("/clients",
  async (IJurisRepository repo) =>
  {
    return Results.Ok(
      await repo.GetClientsAsync());
  }).AllowAnonymous();
```

In this way, you can mix and match authentication and authorization as you like.

Using Minimal APIs without Top-Level Functions

This new change to .NET 6 can come to a shock to many of you. You might not want to change your **Program.cs** to use Top-Level Functions for all your projects, but can you still use Minimal APIs without having to move to Top-Level Functions. If you remember, earlier in the article I mentioned that most of the magic of Minimal APIs comes from the **IEndpointRouteBuilder** interface. Not only does the **WebApplication** class support it, but it's also used in the traditional **Startup** class you may already be using. When you call **UseEndpoints**, the delegate you specify there passes in an **IEndpointRouteBuilder,** which means you can just call **MapGet**:

```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env)
{
  if (env.IsDevelopment())
  {
    app.UseDeveloperExceptionPage();
  }

  app.UseRouting();

  app.UseAuthorization();

  app.UseEndpoints(endpoints =>
  {
    endpoints.MapGet("/clients",
      async (IJurisRepository repo) =>
      {
        return Results.Ok(
          await repo.GetClientsAsync());
      }).AllowAnonymous();
  });
}
```

Although I think that Minimal APIs are most useful for greenfield projects or prototyping projects, you can use them in your existing projects (assuming you've upgraded to .NET 6).

Where Are We?

Hopefully, you've seen here that Minimal APIs are a new way to build your APIs without much of the plumbing and ceremony that are involved with Controller-based APIs. At the same time, I hope you've seen that as complexity increases, Controller-based APIs have benefits as well. I see Minimal APIs as a starting point for creating APIs and, as a project matures, I might move to Controller-based APIs. Although it's very new, I think Minimal APIs are a great way of creating your APIs. The patterns and best practices about how to use them will only get answered in time. I hope you can contribute to that conversation!

If you'd like a copy of the code for this article, please visit: https://github.com/shawnwildermuth/codemag-minimalapis.

Shawn Wildermuth

SPONSORED SIDEBAR:

Get .NET 6 Help for Free

How does a FREE hour-long CODE Consulting virtual meeting with our expert consultants sound? Yes, FREE. No strings. No commitment. No credit cards. Nothing to buy. For more information, visit www.codemag.com/consulting or email us at info@codemag.com.

Simplest Thing Possible: Tasks

Despite all of the time that Tasks and the Task Parallel Library have been in .NET, their capabilities are still underutilized. For the record, Tasks were introduced in .NET 4, released in February 2010, nearly 12 years ago! That's a few lifetimes in the I/T world. For long-time CODE Magazine readers, you may recall my Simplest Thing Possible (STP) series that ran periodically from 2012 to 2016.

John V. Petersen
johnvpetersen@gmail.com
linkedin.com/in/johnvpetersen

Based near Philadelphia, Pennsylvania, John is an attorney, information technology developer, consultant, and author.

Past hits include three issues on Dynamic Lambdas, Promises in JavaScript, NuGet, and SignalR. I still get pinged on the Dynamic Lambda work—an oldie, but a very much relevant goodie!

I was a bit dumbfounded to realize that I never did an STP on Tasks! Better late than never! The STP series had and has one goal: to get you up and running on a .NET feature as quickly as possible so you can take it to the next level for your context and use-cases. As .NET evolves and its history grows, as well as the numerous blog posts with opinions—some good, some not, some dogmatic, some neutral—more than ever, it has become important to cut through the noise. In most cases, the official Microsoft Documentation is the best source of raw, unopinionated information.

Let's get to it!

What Is a Task?

Think of a Task as a unit of work that is to be executed sometime in the **future**. I've added the emphasis on future because that's what a Task is: a future. An oft-asked question in the various online forums (Stack Overflow, etc.) is whether a Task is a promise in the same way promises are implemented in JavaScript. Promises are supported in .NET through the TaskCompletionSource Class, which is beyond the scope of this article.

In the meantime, consider a Task as a future unit of work that may exist on its own or in the context of TaskCompletionSource. When in the context of TaskCompletionSource, a Task participates in fulfilling a promise. A promise doesn't guarantee that the operation is successful. What's guaranteed is that a result of some kind will be returned. In most cases, it's sufficient to implement tasks as independent entities, apart from TaskCompletionSource. A good use case for TaskCompletionSource is in the API context where some kind of result, even if it's an error, must be returned before yielding control back to the caller. In that regard, TaskCompletionSource is a promise in the same way promises are implemented in JavaScript.

Tasks have been a .NET feature since version 4.0, which was released in 2010. Despite being an available feature for over a decade, it remains a somewhat under-utilized feature, giving credence to the notion that sometimes, what's old is new! The same is true with the async/await language feature introduced in C# 5 in 2012.

A Related Concept: Async/Await

Tasks are asynchronous (async). It's impossible to discuss Tasks without discussing the async/await language feature. Implemented in both C# 5 and VB.NET 11 in 2012, async and await have been around for a long time! If you're going to implement tasks, you must necessarily confront asynchronous programming, which is a very different concept than the synchronous programming of the past. Microsoft made this task much easier when it introduced some "syntactic sugar" to make things easier. In other words, the .NET compiler undertakes the heavy lifting to transform your C# or VB.NET into the necessary IL (Intermediate Language) code to support async calls.

> Sometimes, what's old is new!

An async call, as the name implies, means that the code making the call **does not wait** for the result. And yet, the associated keyword to async is await, which can be confusing because it implies that await means that the code waits, which contradicts async programming! Instead, what it means is that the calling code continues its work **while it waits for (or awaits)** the async code to complete.

Before async/await, you needed to implement callbacks, which meant that you had to pass a reference of the function that was to receive the async function's result. Async calls often took a bit to understand if the only world you were familiar with was synchronous calls where the code would wait, and wait, and continue to wait until either the call completed or timed-out. This often made for a very unpleasant user experience because the client, typically a user interface, wouldn't update. The UI and the whole app appeared to be frozen, and then after some time, everything magically came back to life!

Async/await allows for calls to be non-blocking, meaning that the current thread isn't blocked while the task is running in another thread. Hence, as stated previously, the current thread continues its work **while it waits**. The async/await syntax alleviates you of the direct burden of establishing callbacks. Async programming presents you with an opportunity for better performing applications because it results in more efficient use of resources. Underneath the covers, .NET manages the spawning of a new thread in which to carry out the task, making sure that result gets back to your calling code.

The following document list contains resources that you'll want to review. (Note: Be sure to view the docs relevant to the .NET Framework version that you and your team are using!)

- Task Class: https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-1?view=net-5.0
- TaskCompletionSource Class: https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcompletionsource-1?view=net-5.0
- Async keyword (C#): https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async

- Await operator (C#): https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await
- Async keyword (VB.NET): https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/async
- Await operator (VB.NET): https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/operators/await-operator
- The async programming model: https://docs.microsoft.com/en-us/dotnet/csharp/async
- Task-based async programming model: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming

A Simple Async Task and Test Example

The following is a very simple example of implementing a Task in a custom method:

```
public async Task<HttpResponseMessage>
   WebCall(string url =
    "https://www.google.com") {
    using var client = new HttpClient();
    var result = await client.GetAsync(url);
    return result;
}
```

The WebCall method is a very simple wrapper around the HttpClient class, which contains async methods to perform delete, get, patch, post, and put operations. HttpClient is the preferred class to interact with APIs. If you aren't familiar with the HttpClient class, the reference documents are a good source of information and can be found here: https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-5.0.

> The method signature must be marked as async when the method body contains an await statement.

Whenever a method body contains an await statement, the method signature must be marked as async. The other element to note is the return type specified in the signature: Task<HttpResponseMessage>. The return statement itself is of type HttpResponseMessage. Ultimately, the HttpResponseMessage will be returned **in the future** via a Task. There are four simple elements to implement async programming in this first example:

- Specify in the method signature that the return type is wrapped in a task: Task<HttpResponseMessage>.
- Mark the method signature as async.
- Have at least one **awaitable** method call in the method body.
- Return the type specified in the task generic declaration: HttpResponseMessage.

That's all there is to it. Until use cases become more complex, you don't need to be an expert in all the details of async programming to get up and running for most use cases.

The following is a simple XUnit test that calls the WebCall Method:

```
[Fact]
public async void WebCallASyncTest()
{
    var result = await WebCall();
    var content =
       await result.Content
.ReadAsStringAsync();
    Assert.True(!string
     .IsNullOrEmpty(content));
}
```

Because the WebCall method is marked as async, it is **awaitable**. If you wish to test the async operation, the unit test itself must be async because of the requirement to use the await operator. Note that the **void** keyword instead of Task is used. The Task class comes in two flavors: generic and non-generic. For clarity purposes, if you wanted to replace the void keyword with Task, you can do that. Either way, the unit test executes the code under test asynchronously.

What if you wanted to run an awaitable method in such way that it blocked the current thread? In other words, can you run an async method synchronously? You can do it by directly accessing the task's result property, **which itself is an anti-pattern**! That is not to suggest that employing an anti-pattern is the incorrect thing to do. Employing anti-patterns is often necessary based on context.

The following code is the previous test reworked to run async code synchronously:

```
[Fact]
public  void WebCallSyncTestResult()
{
    var result =  WebCall().Result;
    var content =
        result.Content
         .ReadAsStringAsync().Result;
    Assert.True(!string
       .IsNullOrEmpty(content));
}
```

Wrapping Existing Non-Async Code in a Task

Tasks can be very useful in offloading existing workloads to another thread. Perhaps you have an application that requires a call to a process and you'd like the user to still be able to interact with the UI while it runs. If that code runs synchronously, the UI thread will be blocked until it gets a response back from this long running process. The following example employs the static Task.Run method to wrap a call to a legacy method:

```
[Fact]
public async  void TestLegacyCallWithTask() {
var result =
  await Task.Run(
     () => LegacyProcess(
   new string[1] {"1234"})
```

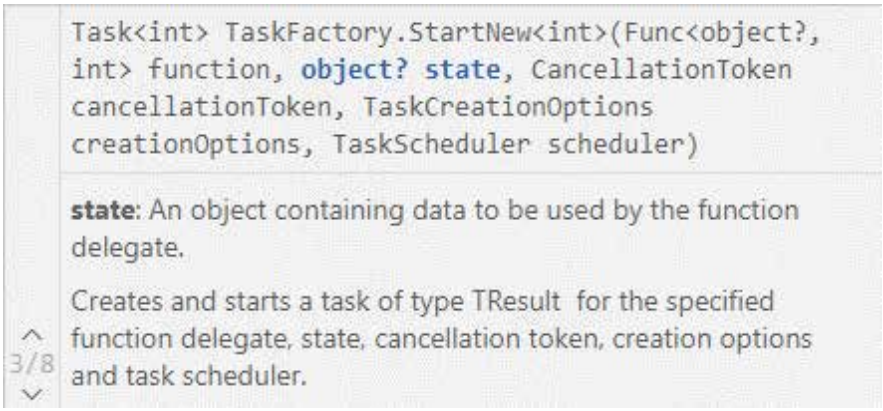


Figure 1: The StartNew() method accepts additional parameters to control how the task is created.

```
  );
  Assert.True(result >= 1);
}
```

The previous example assumes a short running, non-CPU-intensive process. The Run method is a short-hand method for defining and launching a Task in one operation. The Run method causes the task to run on a thread allocated from the default thread pool.

What if it's a long running process? In that case, you'd want to use Task.Factory.StartNew(), which provides more granular control over how the Task gets created by providing access to more parameters. **Figure 1** illustrates the third StartNew method signature with the additional parameters controlling how the Task is created.

> Task.Run is a simpler, shorthand way to create and start a task in one operation.

Task.Run on the other hand, is a simpler, shorthand way to create and start a task in one operation.

If your legacy process is long-running or CPU-intensive, you'll want to use the approach illustrated in the following example:

```
var source = new CancellationTokenSource();
var result =
   await Task.Factory.StartNew<int>(
    () => LegacyProcess(
      new string[1]  {"1234"}),source.Token);
```

Current docs on the Run and StartNew methods may be found here:

- Run(): https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task.run?view=net-5.0
- StartNew(): https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskfactory.startnew?view=net-5.0

For more information on Task creation options, and there are many, that guidance may be found here: https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcreationoptions?view=net-5.0.

Cancellation Tokens

In the previous example, take note of the cancellation token passed as a parameter. This is one of async programming's major benefits, to cancel a long running Task. If the user elects to cancel the task **while it's running**, that can be accomplished **because the current thread is not blocked, allowing user interaction**. A great tutorial on cancelling long-running tasks may be found in Brian Languas's YouTube Video: https://youtu.be/TKc5A3exKBQ.

Conclusion

Tasks and async programming are powerful tools to add to your development toolbox and this article has only scratched the surface. More complex use cases include wrapping multiple tasks together and waiting for all to complete (each running in their own thread .NET's Task and async capabilities are a rich and interesting environment! If you want to see more of this type of content, drop a line to me at CODE Magazine and I'll keep pumping them out. With .NET 5 and VS Code, there's much to distill and demystify.

John V. Petersen

Running Serverless Functions on Kubernetes

Serverless functions are modular pieces of code that respond to a variety of events. It's a cost-efficient way to implement microservices. Developers benefit from this paradigm by focusing on code and shipping a set of functions that are triggered in response to certain events. No server management is required and you can benefit from automated scaling, elastic load balancing, and the "pay-as-you-go" computing model. Kubernetes, on the other hand, provides a set of primitives to run resilient distributed applications using modern container technology. It takes care of autoscaling and automatic failover for your application and it provides deployment patterns and APIs that allow you to automate resource management and provision new workloads. Using Kubernetes requires some infrastructure management overhead and it may seem like a conflict putting serverless and Kubernetes in the same box.

Hear me out. I come at this with a different perspective that may not be evident at the moment.

You could be in a situation where you're only allowed to run applications within a private data center, or you may be using Kubernetes but you'd like to harness the benefits of serverless. There are different open-source platforms, such as Knative and OpenFaaS, that use Kubernetes to abstract the infrastructure from the developer, allowing you to deploy and manage your applications using serverless architecture and patterns.

This article will show you how to run serverless functions using Knative and Kubernetes.

Introduction to Knative

Knative is a set of Kubernetes components that provides serverless capabilities. It provides an event-driven platform that can be used to deploy and run applications and services that can auto-scale based on demand, with out-of-the-box support for monitoring, automatic renewal of TLS certificates, and more.

Knative is used by a lot of companies. In fact, it powers the Google Cloud Run platform, IBM Cloud Code Engine, and Scaleway serverless functions.

The basic deployment unit for Knative is a container that can receive incoming traffic. You give it a container image to run and Knative handles every other component needed to run and scale the application. The deployment and management of the containerized app is handled by one of the core components of Knative, called Knative Serving. Knative Serving is the component in Knative that manages the deployment and rollout of stateless services, plus its networking and autoscaling requirements.

The other core component of Knative is called Knative Eventing. This component provides an abstract way to consume Cloud Events from internal and external sources without writing extra code for different event sources. This article focuses on Knative Serving, but you'll learn about how to use and configure Knative Eventing for different use-cases in a future article.

Development Set Up

In order to install Knative and deploy your application, you'll need a Kubernetes cluster and the following tools installed:

- Docker
- kubectl, the Kubernetes command-line tool
- kn CLI, the CLI for managing Knative application and configuration

Installing Docker

To install Docker, go to the URL https://docs.docker.com/get-docker and download the appropriate binary for your OS.

Installing kubectl

The Kubernetes command-line tool **kubectl** allows you to run commands against Kubernetes clusters. Docker Desktop installs kubectl for you, so if you followed the previous section on installing Docker Desktop, you should already have kubectl installed and you can skip this step. If you don't have kubectl installed, follow the instructions below to install it.

If you're on Linux or macOS, you can install kubectl using Homebrew by running the command **brew install kubectl**. Ensure that the version you installed is up to date by running the command **kubectl version --client**.

If you're on Windows, run the command **curl -LO https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe** to install kubectl, and then add the binary to your **PATH**. Ensure that the version you installed is up to date by running the command **kubectl version --client**. You should have version 1.20.x or v1.21.x because in a future section, you're going to create a server cluster with Kubernetes version 1.21.x.

Installing kn CLI

kn CLI provides a quick and easy interface for creating Knative resources, such as services and event sources, without the need to create or modify YAML files directly. kn also simplifies completion of otherwise complex procedures, such as autoscaling and traffic splitting.

To install kn on macOS or Linux, run the command **brew install kn**.

To install kn on Windows, download and install a stable binary from https://mirror.openshift.com/pub/openshift-v4/clients/serverless/latest. Afterward, add the binary to the system PATH.

Peter Mbanugo
p.mbanugo@yahoo.com
www.pmbanugo.me
twitter.com/p_mbanugo

Peter Mbanugo is a writer and software developer who codes in JavaScript and C#. He is the author of "How to build a serverless app platform on Kubernetes". He has experience working on the Microsoft stack of technologies and also building full-stack applications in JavaScript. He's a co-chair on NodeJS Nigeria, a Twilio Champion, and a contributor to the Knative open-source project.

He's the maker of Hamoni Sync, a real-time state synchronization as a service platform. He works with foobar GmbH as a Senior Software Consultant.

When he isn't coding, he enjoys writing the technical articles that you can find on his website or other publications, such as on Pluralsight and Telerik.

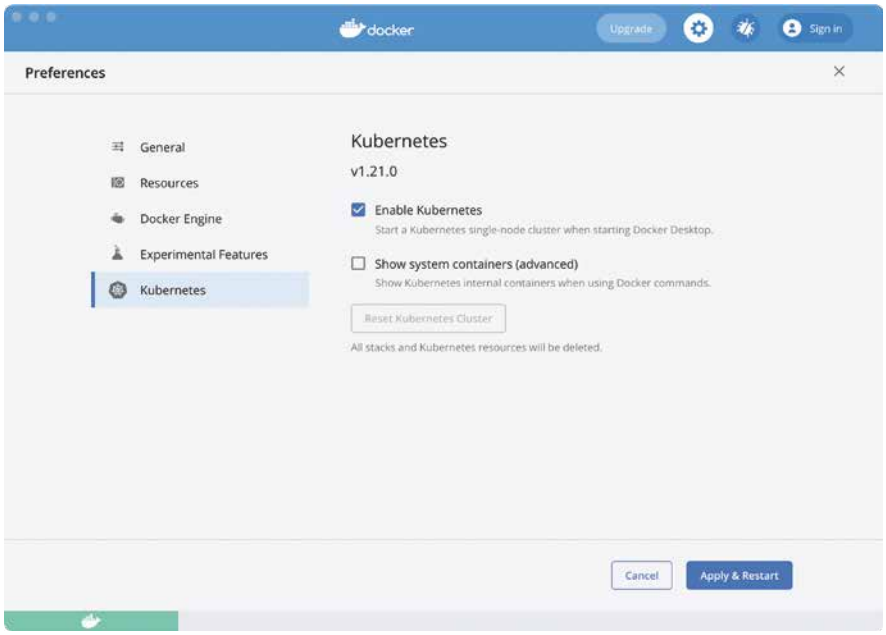


Figure 1: Enable Kubernetes on Docker Desktop

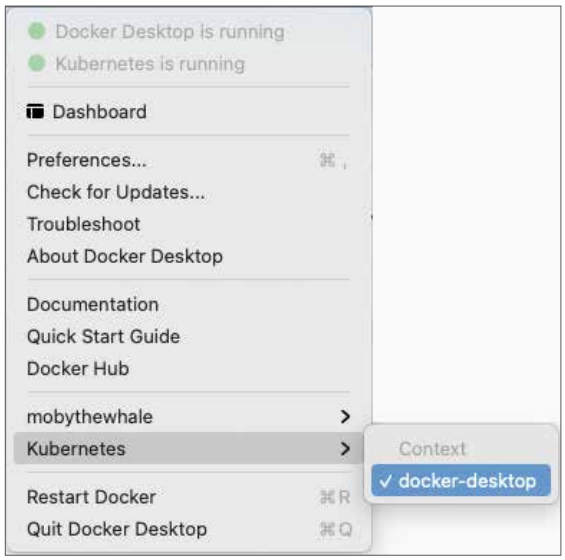


Figure 2: kube context

Creating a Kubernetes Cluster

You need a Kubernetes cluster to run Knative. You can use a local cluster using Docker Desktop or kind.

Create a Cluster with Docker Desktop

Docker Desktop includes a stand-alone Kubernetes server and client. This is a single-node cluster that runs within a Docker container on your local system and should be used only for local testing.

To enable Kubernetes support and install a standalone instance of Kubernetes running as a Docker container, go to **Preferences** > **Kubernetes** and then click **Enable Kubernetes**.

Click **Apply & Restart** to save the settings and then click **Install** to confirm, as shown in **Figure 1**. This instantiates the images required to run the Kubernetes server as containers.

The status of Kubernetes shows in the Docker menu and the context points to docker-desktop, as shown in **Figure 2**.

Create a Cluster with kind

You can also create a cluster using kind, a tool for running local Kubernetes clusters using Docker container nodes. If you have kind installed, you can run the following command to create your kind cluster and set the kubectl context.

```
curl -sL \
https://raw.githubusercontent.com/csantanapr\
/knative-kind/master/01-kind.sh | sh
```

Create a Cluster with DigitalOcean Kubernetes Service

You can also use a managed Kubernetes service like DigitalOcean Kubernetes Service. In order to use DigitalOcean Kubernetes Service (http://digitalocean.com/products/kubernetes/), you need a DigitalOcean account. If you don't have an account, you can create one using my referral link -https://m.do.co/c/257c8259d8ef, which gives you $100 credit to try out different things on DigitalOcean.

You'll create a cluster using **doctl**, the official command-line interface for the DigitalOcean API. After you've created a DigitalOcean account, follow the instructions on docs.digitalocean.com/reference/doctl/how-to/ to install and configure **doctl**.

After you've installed and configured doctl, open your command line application and run the command below in order to create your cluster on DigitalOcean.

```
doctl kubernetes cluster \
create serverless-function \
--region fra1 --size s-2vcpu-4gb \
--count 1
```

Wait for a few minutes for your cluster to be ready. When it's done, you should have a single-node cluster with the name **serverless-function**, in Frankfurt. The size of the node is a computer with two vCPUs, and 4GB RAM. Also, the command you just executed sets the current kubectl context to that of the new cluster.

You can modify the values passed to the **doctl kubernetes cluster create** command. The **--region** flag indicates the cluster region. Run the command **doctl kubernetes options regions** to see possible values that can be used. The computer size to use when creating nodes is specified using the **--size** flag. Run the command **doctl kubernetes options sizes** for a list of possible values. The **--count** flag specifies the number of nodes to create. For prototyping purposes, you created a single-node cluster with two vCPUs and 4GB RAM.

Check that you can connect to your cluster by using kubectl to see the nodes. Run the command **kubectl get nodes**. You should see one node in the list, and the **STATUS** should be **READY**, as shown in **Figure 3**.

Install Knative Serving

Knative Serving manages service deployments, revisions, networking, and scaling. The Knative Serving component

exposes your service via an HTTP URL and has safe defaults for its configurations.

For kind users, follow the instructions below to install Knative Serving.

1. Run the command curl -sL https://raw.githubusercontent.com/csantanapr/knative-kind/master/02-serving.sh | sh to install Knative Serving.
2. When that's done, run the command curl -sL https://raw.githubusercontent.com/csantanapr/knative-kind/master/02-kourier.sh | sh to install and configure Kourier.

For Docker Desktop users, run the command curl -sL https://raw.githubusercontent.com/csantanapr/knative-docker-desktop/main/demo.sh | sh.

Follow the instructions below to install Knative in your DigitalOcean cluster. The same instructions will also work if you use Amazon EKS or Azure Kubernetes Service.

1. Run the following command to specify the version of Knative to install.

```
export KNATIVE_VERSION="0.26.0"
```

2. Run the following commands to install Knative Serving in namespace **knative-serving**.

```
~ kubectl apply -f \
https://github.com/knative/serving/releases/\
download/v$KNATIVE_VERSION/serving-crds.yaml

~ kubectl wait --for=condition=Established \
--all crd

~ kubectl apply -f \
https://github.com/knative/serving/releases/\
download/v$KNATIVE_VERSION/serving-core.yaml

~ kubectl wait pod --timeout=-1s \
--for=condition=Ready -l '!job-name' \
-n knative-serving > /dev/null
```

3. Install Kourier in namespace **kourier-system**.

```
~ kubectl apply -f \
https://github.com/knative/net-kourier/\
releases/download/v0.24.0/kourier.yaml

~ kubectl wait pod \
--timeout=-1s \
--for=condition=Ready \
-l '!job-name' -n kourier-system

~ kubectl wait pod \
--timeout=-1s \
--for=condition=Ready \
-l '!job-name' -n knative-serving
```

4. Run the following command to configure Knative to use Kourier.

```
~ kubectl patch configmap/config-network \
  --namespace knative-serving \
  --type merge \
  --patch \
'{"data":{"ingress.class":\
"kourier.ingress.networking.knative.dev"}}'
```

5. Verify that Knative is Installed properly. All pods should be in **Running** state and the **kourier-ingress** service configured, as shown in **Figure 4**.

```
~ kubectl get pods -n knative-serving
~ kubectl get pods -n kourier-system
~ kubectl get svc  -n kourier-system
```

6. Configure DNS for Knative Serving. You'll use a wildcard DNS service for this exercise. Knative provides a Kubernetes Job called **default-domain** that will only work if the cluster's LoadBalancer Service exposes an IPv4 address or hostname. Run the command below to configure Knative Serving to use sslip.io as the default DNS suffix.

```
~ kubectl apply -f \
https://github.com/knative/serving/releases/\
download/v$KNATIVE_VERSION/\
serving-default-domain.yaml
```

If you want to use your own domain, you'll need to configure your DNS provider. See https://knative.dev/docs/admin/install/serving/install-serving-with-yaml/#configure-dns for instructions on how to do that.

Serverless Functions on Kubernetes

func is an extension of the kn CLI, that enables the development and deployment of platform-agnostic functions as a Knative service on Kubernetes. It comprises of function templates and runtimes and uses Cloud Native Buildpacks to build and publish OCI images of the functions.

To use the CLI, install it using Homebrew by running the command **brew tap knative-sandbox/kn-plugins && brew install func**. If you don't use Homebrew, you can download a pre-built binary from https://github.com/knative-sandbox/kn-plugin-func/releases/tag/v0.18.0, then unzip and add the binary to your PATH.

Functions can be written in Go, Java, JavaScript, Python, and Rust. You're going to create and deploy a serverless function written in JavaScript.

Create a Function Project

To create a new JavaScript function, open your command-line application and run the command **kn func create sample-func --runtime node**. A new directory named **sample-func** will be created and a Node.js function project will be initialized. Other runtimes available are: Go, Python, Quarkus, Rust, Spring Boot, and TypeScript.

The **func.yaml** file contains configuration information for the function. It's used when building and deploying the

```
→  kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
serverless-app-platform-default-pool-899pk    Ready    <none>   53m   v1.21.3
→
```

Figure 3: kubectl get nodes

A Good Book to Read

If you're interested in serverless, Kubernetes, and Knative, you should check out my book "How to build a serverless application platform on Kubernetes. You'll learn how to build a serverless platform that runs on Kubernetes using GitHub, Tekton, Knative, Cloud Native Buildpacks, and other interesting technologies. You can find it on https://books.pmbanugo.me/serverless-app-platform.

```
bash-3.2$ kubectl get pods -n knative-serving
NAME                                    READY   STATUS    RESTARTS   AGE
3scale-kourier-control-54cc54cc58-r5bzt   1/1     Running   0          68m
activator-67656dcbbb-jtcbx              1/1     Running   0          69m
autoscaler-df6856b64-kgpn2              1/1     Running   0          69m
controller-788796f49d-msl66             1/1     Running   0          69m
domain-mapping-65f58c79dc-f4xkd         1/1     Running   0          69m
domainmapping-webhook-cc646465c-s7jn2   1/1     Running   0          69m
webhook-859796bc7-np4fh                 1/1     Running   0          69m
bash-3.2$ kubectl get pods -n kourier-system
NAME                                    READY   STATUS    RESTARTS   AGE
3scale-kourier-gateway-6d8f6b8549-29pv8   1/1     Running   0          68m
bash-3.2$ kubectl get svc  -n kourier-system
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
kourier            LoadBalancer   10.111.204.213  localhost     80:30885/TCP,443:31055/TCP   68m
kourier-internal   ClusterIP      10.101.124.35   <none>        80/TCP                       68m
bash-3.2$
```

Figure 4: Verify installation

```
{
    target: "Web",
    query: { },
    time: "2021-11-24T12:12:51.110Z"
}
```

Figure 5: Function invocation response

function. You can specify the buildpack to use, environment variables, and options to tweak the autoscaling options for the Knative Service. Open the file and update the **envs** field with the value below:

```
- name: TARGET
  value: Web
```

The **index.js** file contains the logic for the function. You can add more files to the project, or install additional dependencies, but your project must include an index.js file that exports a single default function. Let's explore the content of this file.

The index.js file exports the **invoke(context)** function that takes in a single parameter named **context**. The **context** object is an HTTP object containing the HTTP request data such as:

- **httpVersion**: The HTTP version
- **method**: The HTTP request method (only GET or POST supported)
- **query**: The query parameters
- **body**: Contains the request body for a POST request
- **headers**: The HTTP headers sent with the request

The **invoke** function calls the **handlePost** function if it's a POST request, or the **handleGet** function when it's a GET request. The function can return void or any JavaScript type. When a function returns void, and no error is thrown, the caller will receive a 204 No Content response. If you return some value, the value gets serialized and returned to the caller.

Modify the **handleGet** function to return the value from the **TARGET** environment variable, the query paramteter, and date. Open **index.js** and update the **handleGet** function with the function definition below:

```
function handleGet(context) {
  return {
    target: process.env.TARGET,
    query: context.query,
    time: new Date().toJSON(),
  };
}
```

Deploy the Function

To deploy your function to your Kubernetes cluster, use the **deploy** command. Open the terminal and navigate to the function's directory. Run the command **kn func deploy** to deploy the function. Because this is the first time you're deploying the function, you'll be asked for the container registry info for where to publish the image. Enter your registry's information (e.g., docker.io/<username>) and press **ENTER** to continue.

The function will be built and pushed to the registry. After that, it'll deploy the image to Knative and you'll get a URL for the deployed function. Open the URL in a browser to see the returned object. The response you get should be similar to what you see in **Figure 5**.

You can also run the function locally using the command **kn func run**.

Other Useful Commands

You're now familiar with creating and deploying functions to Knative using the build and deploy commands. There are other useful commands that can come in handy when working with functions. You can see the list of commands available using the command **kn func --help**.

The **deploy** command builds and deploys the application. If you want to build an image without deploying it, you can use the **build** command (i.e., **kn func build**). You can pass it the flag **-i <image>** to specify the image, or **-r <registry>** to specify the registry information.

The **kn func info** command can be used to get information about a function (e.g., the URL). The command **kn func list** on the other hand, lists the details of all the functions you have deployed.

To remove a function, you use the **kn func delete <name>** command, replacing **<name>** with the name of the function to delete.

You can see the configured environment variables using the command **kn func config envs**. If you want to add an environment variable, you can use the **kn func config envs add** command. It brings up an interactive prompt to add environment variables to the function configuration. You can remove environment variables as well using **kn func config envs remove**.

What's Next?

Serverless functions are pieces of code that take an HTTP request object and provide a response. With serverless functions, your application is composed of modular functions that respond to events and can be scaled independently. In this article, you learned about Knative and how to run serverless functions on Kubernetes using Knative and the **func** CLI. You can learn more about Knative on knative.dev, and a cheat sheet for the kn CLI is available on cheatsheet.pmbanugo.me/knative-serving.

Peter Mbanugo

(Continued from 74)

a time, it stated that scrum was simple to understand but difficult to master. I believed that when it was written, and I still believe that today, because it's true. The problem with the Scrum Guide is that it no longer states that simple, plain truth.

In the 2020 version, that statement was reduced to "scrum is simple." The 2020 guide also states that Scrum is "purposely incomplete." When the revision history between 2017 and 2020 is reviewed (http://www.scrumguides.org/revisions.html), you can note that, among other things, scrum has become less prescriptive.

> Scrum is simple to understand but difficult to master.

There's also the change in wording from self-managing to self-organizing. Is that a distinction without a difference? Is that plain, simple language that leads to better understanding? No. That's a subjective call. It's one that each team and organization must answer for itself.

This is also not a rant against scrum, whether it's the scrum.org camp or the scrumalliance.org camp. Although there are two different organizations, and they both point to the same Scrum Guide (www.scrumguides.org) and the Agile Manifesto (www.agilemanifesto.org). How each camp describes scrum isn't always consistent. Heck, I've been practicing Agile principles in one form or another for over 20 years and I'm confused. It's for that reason that I've decided to use the Agile Manifesto as my sole basis to define Agile, which bears repeating in its entirety here:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right (processes and tools), we value the items on the left (individuals and interactions) more.

The Agile Manifesto nailed it as far as A): plain, simple language, and B): what could be built from it.

Is Agile alone enough to build software? Of course not. Agile doesn't exist in a vacuum. Software development doesn't exist in a vacuum either. Business doesn't exist in a vacuum. If an organization is to adopt Agile, software development and the business must converge so that they can work consistently toward the same goals and objectives.

What is your software supposed to do? What is it not supposed to do? There are two essential elements, two sides of the software development coin: A): IT, and B): the business. Between the two, there's a chasm that must be spanned. The notion of a Ubiquitous Language, a term coined in the software development context in 2002, was on the right track but missed the mark through all its mechanics and ceremony. And it should be noted that IT serves the business, period.

All software has business requirements. Such requirements are going to be articulated in business terms by the business, as they should be, because it's the business that owns the software and will be the arbiter of whether the software works and consequently, whether it delivers value. It's incumbent on software developers to be the Rosetta Stone to make the translation. We must become experts enough in the business to do that. The business isn't going to be experts in software development.

In my experience, open, honest, and transparent communications is a great place to start. At the beginning of every project, people want the same thing. Agile may or may not be better than Waterfall in a given context; it all depends on your organization. After all, context is a necessary factor in assessing whether language is plain and simple. We must always consider who the audience is. It's an obvious point perhaps, but it's one worth repeating. Employing plain, simple language where individuals and interactions work and collaborate to build, modify, and deliver working software that yields value is, in my opinion, a more straightforward way of articulating the Agile Manifesto's true meaning and intent. As for the Agile Principles, that's up to you and your organization. There is, or at least there should be, a shared goal of getting things accomplished. Only you and your team can assess what those goals and objectives should be. Principles, absent context and what they are supposed to mean, despite what appears to be plain, simple language, are useless.

Every successful project has one thing in common: good people that had a shared goal. Yes, mistakes along the way are made. We never get it right the first or second time in most cases. For us techies, let's avoid tech talk. And if the business is lacking clarity on something, ask for more clarity. If something isn't feasible, explain why not in plain, simple language that conveys the right concerns that the business can act upon. If we don't do that, then how will we understand each other? And if we can't do that, how will we ever build and deliver working software that delivers value?

John V. Petersen

CODE

Jan/Feb 2022
Volume 23 Issue 1

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Contributing Editor
John V. Petersen

Content Editor
Melanie Spiller

Editorial Contributors
Otto Dobretsberger
Jim Duffy
Jeff Etter
Mike Yeager

Writers In This Issue
Bilal Haidar
Sahil Malik
Rod Paddock
Paul D. Sheriff
Mike Yeager
Joydip Kanjilal
Peter Mbanugo
John V. Petersen
Shawn Wildermuth

Technical Reviewers
Markus Egger
Rod Paddock

Production
Friedl Raffeiner Grafik Studio
www.frigraf.it

Graphic Layout
Friedl Raffeiner Grafik Studio in collaboration with onsight (www.onsightdesign.info)

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: American News Company (ANC)
Media Solutions

Subscriptions
Subscription Manager
Colleen Cade
ccade@codemag.com

US subscriptions are US $29.99 for one year. Subscriptions outside the US are US $50.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, e-mail subscriptions@codemag.com.

Subscribe online at
www.codemag.com

CODE Developer Magazine
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379
Phone: 832-717-4445

CODA: On Plain, Simple Language

The Danish physicist and Nobel Laureate Niels Bohr believed that any concept, no matter how complex, should be explainable in plain, simple language. Bohr was instrumental in aiding understanding of Quantum Theory, a very complex subject indeed. He believed that once the heavy lifting was completed as to any theory, the key to advancement was in a collective understanding of what something generally is and its associated benefits. Not everyone was equally an expert in Quantum Theory. And there was only one Niels Bohr. Although he could have a conversation with others in his field on an equal footing, others needed to have some basic understanding of what these complex and abstract ideas could practically accomplish. Institutions and governments funded research, but sanction and funding were ultimately implemented by people; people who weren't on the same intellectual plane as Niels Bohr.

Simple does not mean dumbed-down. Simple means not complex. We can understand, in general terms, how a clock, a car, or a computer works without knowing how to build those things. Think of an elevator pitch and what's required for it to work. If an elevator ride lasts no more than 15 seconds, in that time, to get to the next step, the idea or concept must be expressed in language that anybody can understand. Understanding does not require expertise. If getting to the next step required equal footing on the same expertise, nothing would ever be accomplished. We're just trying to get to the next step, not solve every problem immediately.

What, then, qualifies as "plain, simple language?" That very much depends on your point of view and how that point of view meshes with other points of view. Those points of view must be reconciled for it to seem simple to all parties.

A specific area in software development where a common language between developers and users is in Domain Driven Design (DDD), an approach developed by Eric Evans. In that work, he coined the term Ubiquitous Language. Evans stated:

DDD is a very good book for outlining what the two sides of the software development coin, developers and users, **should** do. If it were only that easy, to just speak the same language, to use common terms that reference the same commonly accepted thing. Bounded contexts and well-defined domain models sound good. Those things, which are DDD elements, are themselves the end-result of some other process. It begs the question of how we, the two sides, build "quality" software.

Quality can be a bit of a subjective, loaded term. In this context, is the software easy to understand? Does it work according to customer's expectations? That's a practical, not a theoretical concern. If the software doesn't work, all the theory in the world won't matter. The art of software development, as I see it, is the proper mix of theory and pragmatism. There's a lot of good theory in the DDD book and other resources, with some of it being useful in a pragmatic sense. It's always important to remember that the value of theory is a function of its application in the real world.

One possible answer to address how to employ plain, simple language is the Agile Manifesto and its 12 Principles (http://www.agilemanifesto.org/principles.html), in which the principles themselves are based on the Agile Manifesto (http://www.agilemanifesto.org). The Agile Manifesto and its 12 Principles are abstract concepts and are arguably written in plain, simple language. That's a subjective call based on a shared point of view.

The one thing I always found interesting about the Agile principles is the relative position of principle numbers One and Seven:

- Principle One: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Principle Seven: Working software is the primary measure of progress.

Why aren't these two principles adjacent to one another? Is that important? The only way software can deliver value to the business is if it's A): delivered, and B): works per the business's requirements. Reading the principles, the two are disconnected. Their relative position to one other is itself important context that aids understanding and thus goes to whether the language is truly "plain, and simple!" This is where having a good editor matters.

Then there's Principle Four: Business people and developers must work together daily throughout the project.

Does this mean that developers aren't or can't be business people? Are these mutually exclusive? What does it mean to "work together?" What about understanding each other?

How about Principle Three: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.

A couple of weeks vs. a couple of months? The term "couple" is more idiomatic English than a precise ordinal measure. Is it two weeks?

The principles declare a preference for "the shorter timescale." Scrum, an agile-based framework, arguably conflicts, or, at the very least, makes ambiguous, whether Principle Three is a bona-fide agile requirement because at the conclusion of each sprint, the aim is to deliver "potentially releasable software." If you don't release software, it isn't delivered. And if it isn't delivered, it isn't delivering any value. Just ask any businessperson who's funding the project with a budget and business objectives! The conflict here is that on one hand, scrum is based on Agile Principles. On the other hand, arguably, scrum, via the Scrum Guide (http://www.scrumguides.org) appears to throw this Agile principle out the window! In other words, as plain and simple as the language appears, when the one thing (the collection of principles) conflicts with the other thing (the manifesto) it's based upon, is the language really plain and simple? To the contrary, the language is ambiguous and becomes less and less useful. In such a case, the gulf between the theory and its practical usefulness increases. Theory, without any sort of practical application in the business context, is useless. In my opinion, Agile has drifted away from its roots in the manifesto, in part because of a lack of the plain, simple language that leads to better understanding.

I've always adhered to one basic principle: When in doubt, go back to basics. This isn't a rant against Agile. But of late, there has been some revolt against Agile and its progeny—scrum, in particular—citing that Agile just doesn't work. The Scrum Guide, where scrum is codified, has been "tinkered" with over the years. Once upon

(Continued on page 73)

CODE
CONSULTING

TAKE AN HOUR ON US!

Does your team lack the technical knowledge or the resources to start new software development projects, or keep existing projects moving forward? CODE Consulting has top-tier developers available to fill in the technical skills and manpower gaps to make your projects successful. With in-depth experience in .NET, .NET Core, web development, Azure, custom apps for iOS and Android and more, CODE Consulting can get your software project back on track.

Contact us today for a free 1-hour consultation to see how we can help you succeed.

codemag.com/OneHourConsulting
832-717-4445 ext. 9 • info@codemag.com

CODE
LEGACY

NEED MORE OF THIS?

Is slow outdated software stealing way too much of your free time? We can help.
We specialize in updating legacy business applications to modern technologies.
CODE Consulting has top-tier developers available with in-depth experience in .NET,
web development, desktop development (WPF), Blazor, Azure, mobile apps, IoT and more.

Contact us today for a complimentary one hour tech consultation. No strings. No commitment. Just CODE.

codemag.com/modernize
832-717-4445 ext. 9 • info@codemag.com