Kafka, .NET MAUI, VS Code, Semantic Kernal

CODE

SEP
OCT
2024

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

CODE 30 YEARS

The Art of Orchestrating Kubernetes

Create Custom Scripting Languages

Explore the Outbox Pattern with Kafka

Secrets of VS Code

09
7 25274 56130 8

CODE
CONSULTING

ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN BENEFIT YOU TODAY?

EXECUTIVE BRIEFINGS

Are you wondering how AI can help your business? Do you worry about privacy or regulatory issues stopping you from using AI to its fullest? We have the answers! Our Executive Briefings provide guidance and concrete advise that help decision makers move forward in this rapidly changing Age of Artificial Intelligence and Copilots!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

AI-SEARCHABLE KNOWLEDGEBASE AND DOCUMENTS

A great first step into the world of Generative Artificial Intelligence, Large Language Models (LLMs), and GPT is to create an AI that provides your staff or clients access to your institutional knowledge, documentation, and data through an AI-searchable knowledgebase. We can help you implement a first system in a matter of days in a fashion that is secure and individualized to each user. Your data remains yours! Answers provided by the AI are grounded in your own information and is thus correct and applicable.

COPILOTS FOR YOUR OWN APPS

Applications without Copilots are now legacy!

But fear not! We can help you build Copilot features into your applications in a secure and integrated fashion.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/ai-services

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

Features

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. 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.

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

The Ultimate Education Destination

ORLANDO 2024

NOVEMBER 17-22, 2024

LOEWS ROYAL PACIFIC RESORT, ORLANDO, FL

Visual Studio LIVE! Data Platform LIVE! Artificial Intelligence LIVE! Cloud & Containers LIVE!

ARCHITECT YOUR SUCCESS: DEV-FOCUSED TRAINING AT LIVE! 360

Live! 360 Tech Con brings the Developer, Data, AI, and other tech communities together for real-world, practical training on a wide range of Microsoft and other products, technologies, and solutions. Our 6 co-located conferences feature expert education, training, knowledge transfer, and networking, enabling you to create your own custom conference choosing from over 200+ sessions and 16 hands-on labs and in-depth workshops to best suit your needs.

TOPICS INCLUDE...

- .NET
- Javascript
- C#
- Python
- Angular
- Blazor
- Copilot
- Azure
- GitHub
- MAUI
- XAML
- SQL
- Power BI
- DevOps
- Open AI
- Langchain
- Kubernetes
- Generative AI
- Entity Framework
- Microsoft Fabric
- Machine Learning
- Microsoft Cloud
- Data Lakehouse, Warehouses

HANDS-ON LABS & WORKSHOPS ON:

- C#
- Python
- Blazor
- SQL
- Generative AI
- GitHub Copilot
- ASP.NET Core & Razor
- Microsoft Fabric
- Mastering Human Factors
- Building .NET Apps/Sites
- Design for Dev Teams
- Azure OpenAI & LLMs

Use promo code: CODEMAG to save $500*

*off standard pricing; terms & conditions apply

Bring the team along with you! Whether you are a developer, IT Pro, DBA, security specialist, or a Data Scientist, Live! 360's 6 co-located events are crafted for (almost) every title! Plus, group discounts are available.

live360events.com/orlando

Potential and Possibility

It's time for another art image from your trusty editor. I like to call this the Pile of Possibility: pencils, pens, paint, Legos and crayons. These are all the tools and materials capable of endless artistic possibilities. The tools and materials of the software developer consist of code, compilers, text editors, and, every so often, a carefully chosen icon. And just like art materials, they're capable of endless possibilities.

I think the realm of the possible is what makes being a software developer so amazing. We take raw ideas, transform them into designs and code, and, with any luck, a functioning application. I love the phone calls from teammates or clients where they say, "Hey Rod we need to do X—how would you tackle this?" "Holy cool idea, Batman!" I answer in my Robin voice. "Let me take a look."

Many years ago, I was having a discussion with a client just like the one above. Here's how it went: "Hey Rod we need to replace our call center scripting app with something more extendable and maintainable. Have any ideas?" My reply: "Oh cool, I just read about a feature in the latest .NET Framework. Here, we can take our C# code and embed it into Microsoft Word documents. This might work." The client replied "That makes sense because our call center managers use Word for this anyhow. Let's give that a try." Music to my ears. I went to work.

I decided to create a small application using legendary developer Ward Cunningham's philosophy: "The Simplest Thing That Can Possibly Work" (https://www.artima.com/articles/the-simplest-thing-that-could-possibly-work). I added a Smart Tag that opened a .NET WinForms with a text field and a button. This form added the contents of the text field to the document as well and then saved that content internally so that it could be retrieved later. That's it: short, sweet, and to the point.

Lo and behold, it worked. I scheduled a call with the client and showed my results. They were happy and we went to work on the real application. This simple proof of concept resulted in us creating a fully functional call-center management system. Call center scripts could be fed to a "Runner" which opened the document, parsed it, and created fully functional applications generated by end-users with zero coding capabilities. This is the power of possibility realized. This is what makes our job fun and interesting.

I'm not sure how or why I paid attention to this new tool set when it was created but I'm happy I did. My thought process was probably something like: "This is a pretty cool feature. I wonder where it would be useful?" I saw potential and was given the possibility to create something cool.

Figure 1: All the creative tools I need.

This philosophy of looking for potential has served me well and continues to this day. That leads me to the current state of the technological ecosystem. We're currently in a realm of huge potential and possibility. I'm talking about what everyone's talking about: artificial intelligence, machine learning, and LLMs. We've all been using these tools for some time, but recently, the power and accessibility of these tools has grown exponentially, and it's up to us to take advantage of the possibilities that these technologies give us.

I'm not sure how we'll take advantage of these technologies in our day-to-day development but here at CODE Magazine, we'll be researching these ideas and developing content to help you see the potential and possibilities of them in your daily development. We'll try to help answer questions like:

- Where are LLMs useful?
- How do I use my own data to populate an AI model?
- What is machine learning and how do I use it?

Of course, there are multitudes of questions that seem to arise daily and we'll try to roll with them as they come up.

We live in exciting times and these tools seem to present endless possibilities, much like creation of the internet and mobile phones presented possibilities never seen previously.

I'm going to conclude with one of my favorite sayings, which shows us how we need to keep potential and possibility in our hearts:

> Every child is an artist.
> The problem is staying an artist when you grow up.
>
> *Pablo Picasso*

Never grow up. See what happens.

Rod Paddock
CODE

CODE: 10 Years Ago

In this article, I'm continuing our journey through the last 30 years of anything related to CODE. We're starting to arrive in a reasonably modern era. In fact, when I researched some of the background information for this article, I was continuously surprised that some of these things were a decade ago. Some of the topics discussed in this article seem like they happened

"just the other day." I even still have some of the games and movies from 10 years ago on my "to be watched/played" list. Wow!

Operating Systems

Let's start our exploratory trip through time with the generation of operating systems in use. For Microsoft, things did not go particularly well a decade ago. In 2012, Microsoft released Windows 8, and oh boy! Where do I start? I remember attending BUILD 2012 at Microsoft. And when I say, "at Microsoft," I really mean "on the Microsoft campus in Redmond," because that particular event was held on location at Microsoft headquarters. As far as I know, this was the only event of this size (over 5,000 people) that was ever held on campus and there really wasn't enough meeting space for it. Microsoft put up tents to house part of the event, including all the large sessions, such as the keynotes.

The event itself was highly anticipated, because Microsoft had already been on the back foot for a while, starting to slip badly in operating system market share (both Apple and Linux had turned out to be feisty competitors) and having completely lost its previously strong position in the mobile phone space (with Apple and Android dominating). Therefore, the release of Windows 8 and Windows Phone 8 had been anticipated with both excitement and trepidation.

There was trepidation because, during the beta phases of Windows 8, the new version of Microsoft's flagship product hadn't been received well. Windows 7 had been a great operating system that was well received on the desktop platform, but Apple's iPad had been eating Microsoft's lunch as more and more desktop users were successfully lured into the tablet space. Microsoft had nothing to put up against it. (Neither did Google, as Android on tablets was a niche market and remains so to this day). Previously, Microsoft had Tablet PCs, which were very interesting devices, but never made it quite to the level of polish that the iPad had to offer. Although some laptops had tablet features, they were clearly a PC with some tablet capabilities, rather than a highly polished tablet offering.

Windows 8 was Steven Sinofsky's idea of taking the fight to Apple. (Steven Sinofsky, a Microsoft veteran, oversaw Windows 8 within Microsoft.) The idea was to change the user experience to be more touch-friendly, which included a simplified app environment, full-screen operations, swipe gestures, and many similar concepts. All good ideas, it would seem. However, the implementation was such that it still couldn't compete with Apple's offerings (at the end of the day, Windows still ran PCs and not tablets) while at the same time completely ruining the desktop experience. Not only was the start menu overly dumbed down for desktop users, but to the great befuddlement of just about anyone, Microsoft had removed the Start-button that had become synonymous with Windows. This led to even experienced users sitting in front of their computer not knowing how to launch applications or bring up the Start screen (**Figure 1**).

Microsoft realized that most Windows applications were not easily usable in touch-mode. Therefore, a new UI paradigm with new applications was needed. Microsoft's answer was WinRT apps. These applications were simplified apps that were reasonably operable in tablet mode, but mostly laughable for desktop apps. This, in itself, would not have been a big issue if WinRT apps had simply been available for tablet users as an added benefit, or perhaps secondary modes of conventional applications. Instead, Microsoft made the questionable decision to try to shove these apps out to all users equally, which was not well re-

Figure 1: The much-hated Windows 8 Start screen

Markus Egger
megger@codemag.com

Markus, the dynamic founder of CODE Group and CODE Magazine's publisher, is a celebrated Microsoft RD (Regional Director) and multi-award-winning MVP (Most Valuable Professional). A prolific coder, he's influenced and advised top Fortune 500 companies and has been a Microsoft contractor, including on the Visual Studio team. Globally recognized as a speaker and author, Markus's expertise spans Artificial Intelligence (AI), .NET, client-side web, and cloud development, focusing on user interfaces, productivity, and maintainable systems. Away from work, he's an enthusiastic windsurfer, scuba diver, ice hockey player, golfer, and globetrotter, who loves gaming on PC or Xbox during rainy days.

ceived. These apps were just too simple. They were always in full-screen mode. Many users quipped that Windows 8 should have really been called "Window 8" (singular) because only one window could be opened at a time.

In short: WinRT apps were not a success and most companies that invested into WinRT development gave up on those efforts sooner or later, often having to write off a great deal of investment in the process.

Windows App Store

Microsoft had more tricks up its sleeve. A major new feature in Windows 8 was the new Windows App Store. After all, Apple, Google, and even Valve, had made their stores popular and successful, and Microsoft had no equivalent offering, a situation that was bizarre, to say the least. After all, the Windows ecosystem represents the largest operating system platform in the world, yet there was no standardized way to sell and deploy applications for it. Low-hanging fruit, one would think, and Microsoft seemed to agree and introduced the first built-in Windows App Store in Windows 8 (**Figure 2**).

Unfortunately, the list of odd decisions made around Windows 8 continued in the App Store. It was hard to find and navigate. (The search feature could only be brought up with a swipe-gesture or by hitting Windows-Q to bring up the "Charms Bar" and then picking the external search-feature there, to name just one strange interface choice that can be used for "how to not design a UI" training classes.) It's hard to describe how awkward all of this

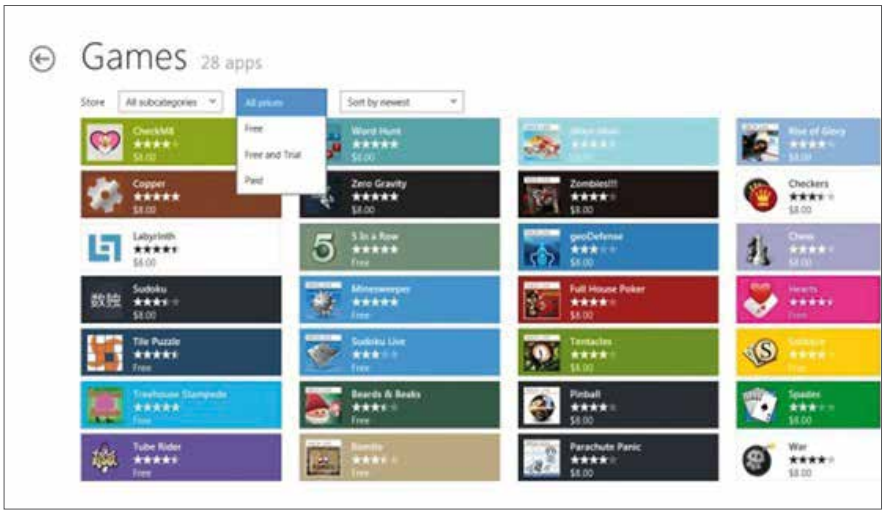

Figure 2: The Windows 8 App Store

Figure 3: Windows Phone 8.1

was. But worst of all, the Windows App Store only supported the new WinRT apps, which very few people had any interest in. This seemed a strange decision to me at the time, and it seems even stranger now. Microsoft made an app store for the world's largest platform yet only made it available to this new type of app, completely ignoring the millions of existing apps that could have been deployed through this system to literally billions of users. Even Microsoft's own applications, such as the desktop version of Office, were not available.

It truly was a strange time. And that gets us right back to the BUILD conference mentioned at the top of this article: Here we were, sitting in a huge tent on the Microsoft campus, waiting for Steven Sinofsky's keynote, hoping Microsoft would pull a rabbit out of its hat. Instead, we learned just before the start of the event that Sinofsky had been removed from his position and was going to leave Microsoft. It made for an awkward keynote delivered by someone else (probably Steven Balmer, although I don't even remember), and it left many of us distraught and worried. Perhaps saying that the situation was "strange" is quite an understatement. Without looking up the exact numbers, I'd guess that Steven Sinofsky presided over one of the largest destructions of brand value in the history of capitalism.

Microsoft managed to put out a *slightly* better version with Windows 8.1 a year later, but the damage was done, and it took a long time to recover from it.

Windows Phone

Luckily, it wasn't all about Windows 8. After all, Microsoft was also about to release Windows Phone 8 and Windows Phone 8.1 a few months later (**Figure 3**). Microsoft had been an early leader in mobile devices with products like Windows CE and Windows Mobile, but had fallen far behind in the mobile race, largely due to Steve Balmer's misguided belief that business phones needed a physical keyboard. (The same assumption also eventually sunk RIM's BlackBerry product line). Meanwhile, Apple had done an excellent job establishing the iPhone and Google provided a popular alternative with Android. Microsoft was nowhere and Windows Phone 8 was to change that.

In hindsight, the Windows Phone platform was actually quite good on a technical level. The tile-based interface worked reasonably well on a phone and provided a number of features that neither Apple nor Google had. On top of that, the soft keyboard the operating system provided was clearly the best keyboard available (although not comparable to modern, AI-driven keyboards). However, the effort ultimately proved to be too little, too late. Although the other two platforms had a very healthy application ecosystem, Microsoft started from scratch. Efforts to entice the most popular app makers to create apps for the Windows Phone platform ended up with little to show for it. Yes, many of the most important apps were available on Windows Phone, but they were clearly low-budget productions that ticked a lot of boxes, but just weren't as good as on other platforms, and users noticed that they weren't getting quite the same experience on their Windows Phone as their friends with iPhones and Android Phones got. Especially once you moved past the most obvious big-hitter apps, it was clear that most smaller app offerings were only available for iPhone and Android, and Windows Phone users often found themselves out of luck.

It was a shame, but in hindsight, it was an unwinnable proposition. The commitment and financial investment required to be competitive was simply too large, even for Microsoft. One hope was that the platform could provide features that would turn the phone into more of a real computer, with the capability to hook the phone to an external monitor, keyboard, and mouse, never went anywhere. I liked the idea of having a full computer in my pocket (after all, the performance of those devices was similar to recent computers at the time) but I can only speculate that Microsoft didn't want the phone to cannibalize its desktop operating system.

And let's not even dwell on what was the Microsoft Kin, the "phone aimed at 15-30 year old users." It boggles the mind how that could have ever been considered a coherent target audience. (My 30-year-old self was rather a bit different from my 15-year-old self, it seems.) Microsoft sunk billions into that endeavor that was so utterly devoid of success, they pulled the plug on the product within two months. If you don't remember the Kin, don't fret! You didn't miss anything.

Microsoft vs. the Rest of the World

Meanwhile, Microsoft's competitors were having a field-day. Apple kept pushing full speed ahead and enjoyed major success with three different product lines (among smaller additional ones): Mac, iPhone, and iPad. iOS 7 was a major overhaul that simplified the user interface

Figure 4: Mass Effect 3 box art

of the phone in a way that's still familiar to iOS users today. Mac OS X Mavericks was a very popular version of Mac OS X. Phones and iPads were still improving their hardware platforms with meaningful upgrades. Similarly, Google's Android OS released version 4.x (Ice Cream Sandwich, Jelly Bean, and KitKat… yes, those were really the names) that rapidly iterated on performance and graphics improvements, better graphics, and many new features.

The app stores on these platforms had already grown to feature an enormous number of applications and established a world of mobile devices that we still enjoy today.

On the server, Linux continued to be popular. We weren't quite there yet, but Microsoft would soon announce that Linux was fully supported on Microsoft Azure Cloud, a development few saw coming at the time. However, this was the start of a new way of approaching the world for Microsoft, based on the leadership of Satya Nadella, the new Microsoft CEO as of 2014. Nadella had succeeded Steven Balmer, the polarizing Bill Gates successor and CEO of Microsoft, who probably had just made a few too many mistakes around Windows and Windows Phone. Where history does not appear to be kind to Balmer and Sinofsky, Nadella would go on to turn Microsoft around and back to the success story it previously was. Satya Nadella is still at the helm of Microsoft today and is highly regarded and respected in this role.

Software Development and Other Tech Trends

All of the technologies and goings-on described above had a significant impact on the world of software development. Microsoft enabled developers to write apps for the ill-fated WinRT platform. However, because much of this had no longevity, it had little overall impact in the long run. Other important technologies, such as ASP.NET MVC, were stable and had been around for a while. Ignoring the short-lived tech, I look back at this as a time of relative stability for Microsoft developers, even though it may not have seemed like it at the time. Even on other platforms, that seems to be true. Apple, for instance, hadn't quite released its new Swift programming language yet (this happened in 2014), so most apps were still built on the awfully outdated Objective-C language. Also, web developers will likely recognize frameworks such as Angular that were already in use back then.

With all that said, and with all the lack of success around Windows 8 and Windows Phone, Microsoft also had some initiatives that were highly successful. It had become obvious that the Cloud was here to stay, and that Microsoft was a key player in that together with Amazon. Microsoft was starting to realize that a successful cloud platform couldn't exclusively focus on Windows and that Linux was a pretty good way to run web apps. And why wouldn't Microsoft want to make money that way, if many developers enjoyed that approach?

This had far-reaching implications. For instance, the .NET platform was exclusively designed for Windows. Furthermore, it was designed for a world of on-premises servers

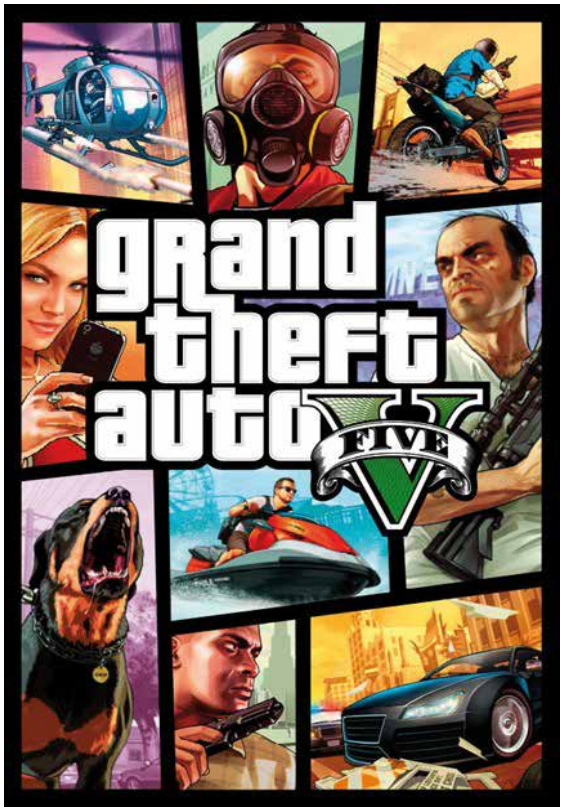

Figure 5: Grand Theft Auto V box art

and desktop applications and not for data center operations and microservices. Therefore, ideas had started to germinate around creating versions of .NET that could run on Linux servers and in much smaller and more efficient packages. These ideas took a while to make it to market and were nowhere to be seen yet (at least in public), but they would later materialize as .NET Core 3.1 (in 2016).

A decade ago is also when we started seeing some interesting tech developments, such as the rise of Big Data and modern machine learning setups. We also saw the introduction of wearable devices. Oculus had just shipped its first Oculus Rift Development Kit, and Apple wasn't too far from releasing the first Apple Watch (not the iWatch, as many had assumed it would be called). IoT (Internet of Things) had grown in importance and we saw the first super-inexpensive devices, such as the first Raspberry Pi.

Politics, Music, Movies, and Games

What else was going on ten years ago? In politics, Barack Obama had just been re-elected for a second term, and Edward Snowden had been chased to Russia, after disclosing what was going on inside the NSA. Pope Francis was elected. The war in Syria was going on, with ISIS starting to emerge. And Russia annexed Crimea from Ukraine. The Arab Spring blew through much of the Arab world, leading to wide-spread hope for modernization. In hindsight, not much of it materialized.

On a happier note, the music industry had stabilized from the earlier Napster turmoil. Taylor Swift released her album "Red" and "1989" relatively soon thereafter. Yes, she already was a big deal then. I don't recall this period as one that produced a ton of classics (some might disagree), although who would forget "Gangnam Style," which became an absolute viral phenomenon. I just watched the original Gangnam Style video, and it still makes my foot tap and smile at the absurdity of it all <g>.

Meanwhile, the gaming industry kept churning out one timeless classic after another. *Diabolo III*, *Mass Effect 3* (**Figure 4**), *Borderlands 2*, *Grand Theft Auto V* (**Figure 5**), *BioShock Infinite*, *Destiny*, *Dragon Age: Inquisition*—do I need to go on? Wow! Just wow! Some of my fondest gaming memories are in these games right there. Personally, I spent far more hours than I'm willing to admit in the fictitious game worlds of Mass Effect, GTAV, and Dragon Age. I would call it the "golden age of gaming," except there were so many other golden ages of gaming too. My hat's off for what the gaming industry has been able to do in the entire 30 years I cover in this series of articles.

A lot of my gaming happened on the PC. Yes, I had an Xbox One (**Figure 6**), a good device, although I liked the earlier Xbox 360 better for its time, and PlayStation 4 was certainly an awesome device. Wii U had some interesting stuff for a somewhat niche market. Nevertheless, the PC had shown time and time again how much staying power it had, and many more gaming performance enthusiasts could extract from that set up, a statement that's been true in pretty much every decade that this 30-year journey covers, and it's true to this day. Nevertheless, the ease of use of gaming consoles should also not be overlooked in pushing the gaming industry forward.

Figure 6: The original Xbox One

This exploration of the past three decades in everything connected to CODE is advancing into what could be considered the contemporary era. In preparing for this piece, I found myself repeatedly astonished to realize that certain events took place a decade ago. Many subjects covered here feel as though they occurred only recently.

Although I don't miss Windows 8 or the ill-fated WinRT application platform, I do feel oddly compelled to start up some of those "old" games. Perhaps I'll do so before I start writing the next installment of this series of articles, which will bring us to just five short years ago. Until then!

Markus Egger
CODE

CODE 30 YEARS

Career Development and Staffing reinvented

You think great talent and cool positions only exist in Silicon Valley? Think again!

CODE Staffing is a high-tech, career development, and - for the lack of a better term - "staffing company," that provides a modern workplace with the number one goal being the development of talent and careers. CODE Staffing was started to disrupt the traditional staffing industry. We provide an environment that attracts top talent. We offer unrivaled benefits and a cool workplace culture with unlimited upward career development opportunities.

Many companies think they cannot provide a work environment that rivals the "cool Silicon Valley startups," and many employees think they won't be able to find such a workplace outside "The Valley." We have changed that equation!

Our Mission

CODE Staffing delivers unparalleled contingent technical staffing solutions to our clients. We empower their businesses with the talent and expertise they need to thrive. We focus on long-lasting partnerships based on trust, transparency, collaboration and by providing access to our extensive network of industry connections.

We provide a great environment for our employees so that we minimize churn for our clients. Our rigorous focus on training and regular meetings with clients' executives ensure that we understand their technological road map and guarantees that we provide the best people for their needs.

At CODE Staffing, we're committed to delivering exceptional value to our clients, providing a fulfilling work experience for our employees, and building long-lasting partnerships within our community.

Our Vision

To achieve our mission, we focus on two core groups: our clients and our people.

For Our Clients

Our business philosophy is centered on building strong, long-lasting partnerships with our clients, rather than pursuing a large number of short-term engagements. By fostering close relationships with the organizations we serve, we aim to become a trusted and integral part of their operations, and to support their growth and success over the short and long term. We are committed to working collaboratively and transparently with our clients to enable their teams to deliver continued value to their businesses.

Our clients want to be able to focus on their business, not on managing multiple vendors for staffing. Here are some of the ways we help our clients build better software...

We act as a single vendor that knows our clients' business.

Prioritizing a few key accounts rather than a multitude of small accounts allows us to provide personalized services tailored to each of our clients. Our team works closely with our clients' organizations to gain a thorough understanding of their intricacies and to better grasp their overall goals and objectives.

We act as trusted industry advisors to our clients.

Our executive team at CODE is a group of technologists who have owned long-standing, successful businesses. They have experience growing technology teams to thousands of people and have learned that the only way to succeed is to invest in our own employees. Every key client is assigned a member of our executive team as their contact – with regular quarterly meetings to understand where they are heading – both as a business and with technology. This approach enables us to ensure that our staff receives training in the technologies that our clients are moving towards, ensuring that we are always prepared to meet their needs.

We are committed to training and adopting new technologies in a straightforward manner.

Our dedication to keeping our staff well-trained and informed on the latest technologies ensures they are fully equipped to tackle any project for our clients. Our commitment to training paired with our vast experience in the field gives our team the confidence they need to excel in every client engagement while staying up to date on the latest technology.

We leverage CODE Training, a division of the CODE group, to train not only our employees in upcoming technologies, but we make the training available to our clients' personnel as well. By doing so, we enable their personnel to concentrate on their business operations, while we take charge of training both our own and our clients' employees in technology.

We provide a clear legal differentiation between clients' employees and contractors, offering more flexibility.

The contractors we provide are our employees – with salary and benefits, as well as regular training. CODE takes the heavy lifting out of managing (sometimes hundreds of) contractors, leaving our clients more time to focus on their projects and growing their business.

We create a system to keep contractors with our clients for the long term.

By employing great developers and providing ownership via profit sharing, CODE Staffing employees are incentivized to stay with us and our clients for the long run. Focusing on training and the long-term investment in top talent not only helps the organizations we work with achieve sustained success but ensures that institutional knowledge doesn't regularly disappear.

For Our People

CODE Staffing is centered around creating an exceptional workplace environment that attracts and retains top talent, enabling us to provide our clients with a stable and reliable team. We prioritize fulfilling our employees' essential needs by providing engaging work, competitive compensation, a sense of ownership and pride, and continuous opportunities for growth in both their technical and interpersonal skills.

Similar to our business philosophy, we prioritize long-term investment in our staff over filling short-term positions. We are committed to developing the full potential of our employees and supporting their career growth over time. To this end, we provide career mentorship opportunities, generous salaries and benefits, a profit-sharing plan that incentivizes everyone to work toward a common goal, and continuous comprehensive training in the latest technologies as well as in the industries our clients are focused on.

At CODE Staffing, we believe that happy employees lead to happy clients. By prioritizing employee satisfaction and delivering outstanding results for our clients, we can build strong, lasting partnerships based on trust, transparency, and exceptional service.

CODE

More VS Code Tips

In my previous article in CODE Magazine (https://www.codemag.com/Article/2408031/VS-Code-Tips), I shared some of my favorite tips when working with VS Code. As I wrote the article, I realized that I was just brain dumping tidbits I've gathered over many years of using VS Code. The list seemed endless. But what about some larger chunks of tips? Things that every developer uses and are built into VS Code that can supercharge your development? I thought it might be nice to take a logical break and come back with those larger chunks of items.

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.

Git

These days, if you're a developer, you most likely use Git. It has sort of become the de facto source control mechanism, whether it's GitHub, or Azure DevOps, or anything else, chances are you use the Git command quite a bit. It's therefore natural that VS Code offers some fabulous Git integration.

Right out of the box, VS Code has some Git features built into it. To demonstrate many of these features, I'll use the source code for MSAL Python hosted at https://github.com/AzureAD/microsoft-authentication-library-for-python. But feel free to use any moderately realistic Git-based repo.

VS Code's Git integration can be accessed by clicking on the sidebar button shown in **Figure 1**.

Figure 1: VS Code's out of the box Git integration

You can click on that button and VS Code prompts you to either clone a repo or open a folder. Either way, you can work with Git-based repos right inside VS Code. For now, I'll clone the MSAL python repo and open it in VS Code. I made a minor modification to the README.md file and added a file, and immediately VS Code's Git integration shows me that I have changes. In fact, right through VS Code, I can perform common functions, such as staging my changes, reverting my changes, or committing the changes, and give it a decent commit message. I can also click on individual files and view a diff. Additionally, each file shows me a nomenclature showing whether a file is modified, untracked, or deleted. You can see in **Figure 2** that I have an untracked and a modified file.

But we know that Git is much more powerful than this basic stuff. If you click the triple dots on the corner of the source control window, you can see a pretty interesting menu. This can be seen in **Figure 3.**

It appears that I've made this change in the dev branch. This is probably something I shouldn't have done. But I don't want to lose all the work I've done. What I should probably do here is stash my changes, branch, and then apply stash. Let's see if I can achieve this through VS Code. To do it, I simply went to the "Stash" menu and created a stash including unstaged. Then I went to branch and created a branch. Finally, I applied my stash right through the menu. If you're curious, you can choose to see the Git output using the "Show Git Output" menu item at the bottom of **Figure 3**, and it'll show you exactly the Git commands being executed behind the scenes. I find VS Code's Git commands to be extremely chatty, but sometimes they can be helpful in figuring out the syntax of a particularly wiry command.

There are many other Git capabilities built right inside VS Code. For instance, it has a pretty good diffing tool built right into it. I'll talk more about diffing later, but all the diff capabilities of VS Code are extremely useful when working with a Git repo. For instance, it lets you do a three-way merge editor where it shows you the incoming (from the server) version on the left, your version on the right, and the merged version at the bottom. It also lets you do merge conflict resolution. When you do a push to a repo, and the version on the server has changed since you last pushed, you get a merge conflict. This is where Git cannot resolve things automatically for you, so it must ask you what changes are still legit. Git splinters your code with weird text that looks like **<<<<<<<<< HEAD** or **>>>>>>>>> theirs**, which can become very difficult to fix on a text editor. Although GitHub has a pretty nice resolve conflicts facility built right into the web-based UI, VS Code also neatly highlights all such changes for you with convenient buttons for which change to accept.

VS Code also has a pretty nice timeline view, which is hidden by default. To view it, press CMD_P, and "> Focus on timeline". This view for the currently open file can be seen in **Figure 4.**

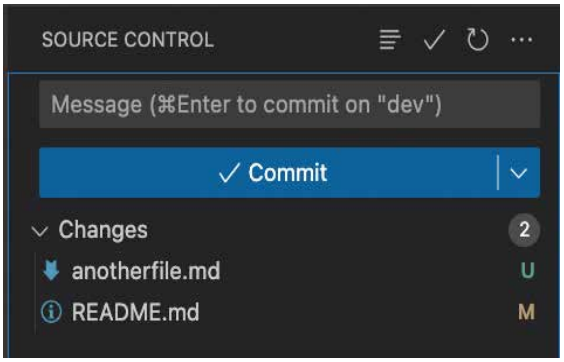

Figure 2: VS Code's Git view

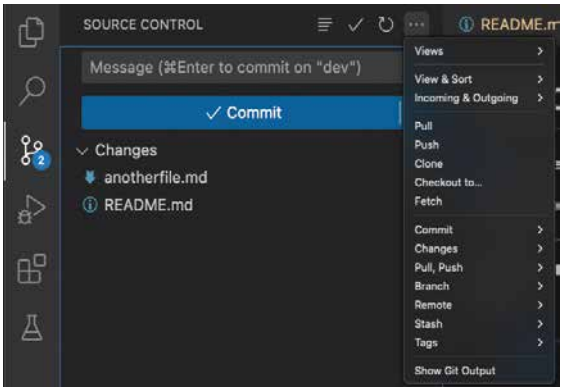

Figure 3: Additional Git features

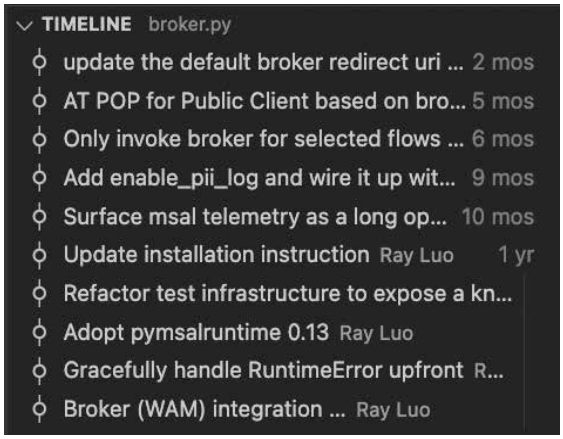

Figure 4: Timeline view

I find the timeline view particularly useful. For instance, I can click on any commit, and view a nice diff view of the change in that particular commit. I can, in fact, right-click on any commit and select for **compare**, and then compare with **selected** to compare any two commits.

Frequently, I look at other people's code, trying to figure out what the heck were they thinking when they wrote this. It's really nice to hover over any particular commit. Simply hoverinver over any commit and the tooltip shows me the developer's comments, which gives me some idea of why the developer made the change. Or I can select a particular commit and click the "View commit" button on the right to get a full snapshot of the commit across all files included in a given commit, without ever leaving VS Code.

When working with Git, frequently Git asks you to provide input. Usually, Git uses VIM as its default editor, which is a great text-based editor, once you master all the shortcuts. I never have to restart my Mac unless I have to exit vim. Well, not anymore. I can tell Git to use VS Code as my default editor by issuing the following command on terminal:

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

VS Code offers great Git integration right out of the box. But a particular VS Code extension I really like when working with Git repos is GitLens. It really supercharges VS Code's capabilities when working with Git.

Once the extension is installed, you'll start seeing Git intelligence right inside your code. For instance, I opened broker.py and placed my cursor on a particular line, and look what it shows me in **Figure 5**.

Every single line now shows you who wrote that line and when. A simple tooltip tells you why that line of code was written. For instance, that particular line was written in a certain commit because Ray Luo wanted to disable SSH cert when using brokers.

You can also choose to view the commit graph in a visual manner of any repo, so you can get an idea what was released and merged at what time and why. This can be seen in **Figure 6**.

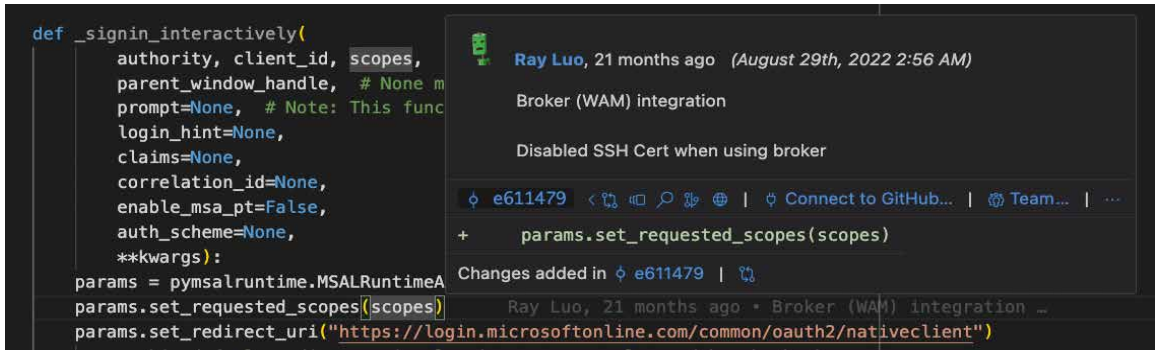

Figure 5: Gitlens integration inside VS Code

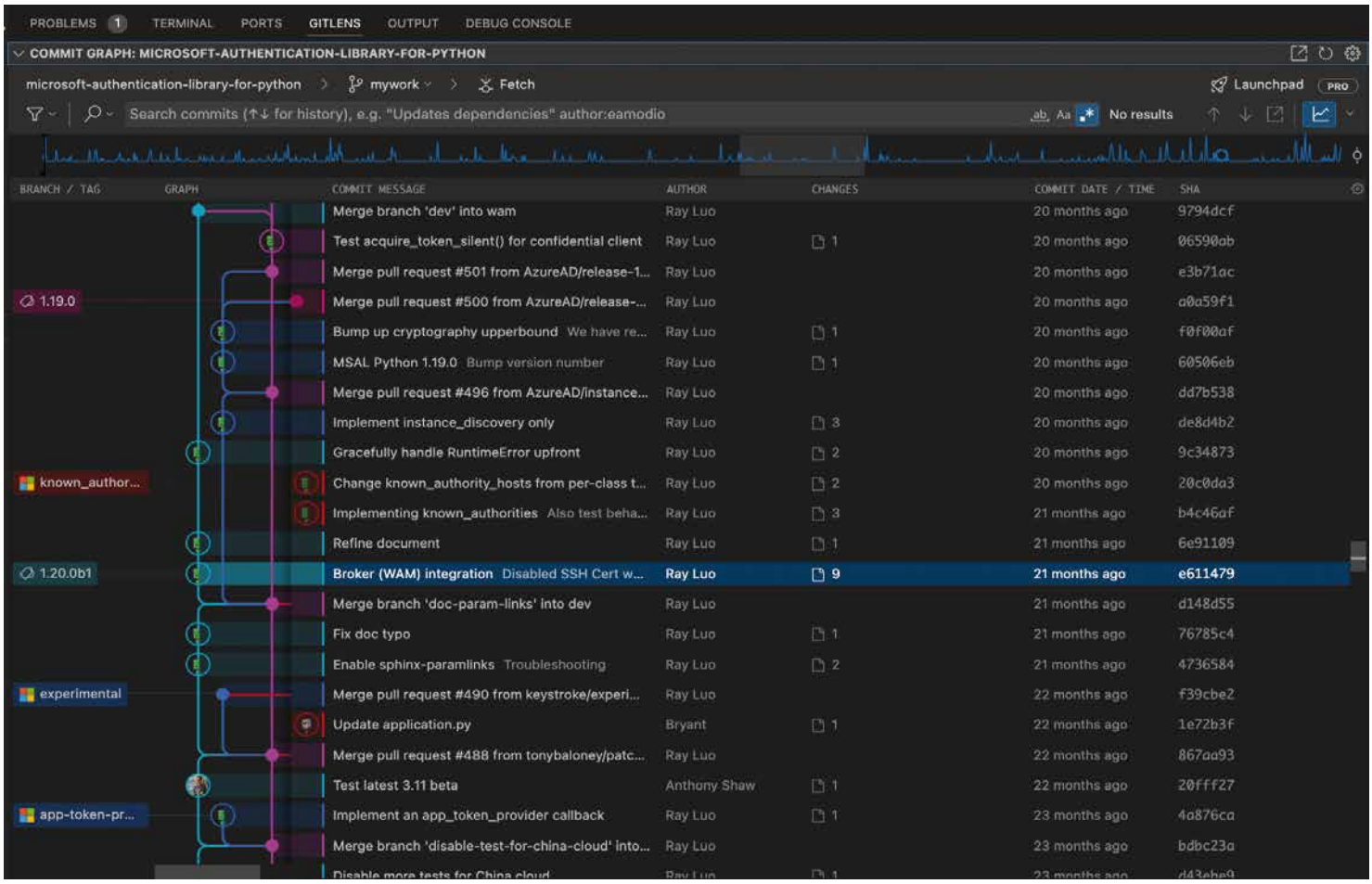

Figure 6: Commit graph in GitLens

Also, note that right through the UI, I can fetch a particular commit and replicate the world as it existed at that time. Sure, I could do that through Git commands too, but this is quicker and easier.

There are a lot of amazing features here, but let's switch gears a bit now and talk a bit about terminal integration in VS Code.

Terminal

In any modern programming language, you'll find yourself on the command line sooner or later. This isn't an unreasonable expectation because these days we develop on a *nix or pretend *nix environment, with the eventual goal of running things in a very lightweight container somewhere, which is typically Linux or similar. Some of that mentality bleeds into our programming lifecycle.

I really like this lightweight and super customizable way of coding. Scripting also ensures reusability, automation, among other things. It's not a surprise therefore that VS Code has tight integration with your terminal. Right inside of VS Code, you can press the **CTRL_`** keyboard shortcut to launch terminal. You can have multiple terminals right inside of VS Code using the **CTRL_SHIFT_`** keyboard

shortcut, or you can launch your actual terminal outside of VS Code using the **SHIFT_COMMAND_C** shortcut. Either way, the terminal of your choice is just a shortcut key away.

Personally, I like to have terminal open within VS Code. I have mentally replaced ALT_TAB with **CTRL_`** when thinking terminal. Try this out yourself. Open VS Code, open any code file, and press **CTRL_`.** It immediately takes you to terminal. Your cursor is now on terminal so you can start typing shell commands. Now, to go back to your code file, press **CTRL_`** again, and you're back in code and the terminal window is hidden. Don't worry, the terminal is still there. It's just hidden. You can maintain multiple such terminal windows right within VS Code and go back and forth between code and terminal using **CTRL_`**.

Another reason I prefer having terminal right inside VS Code is because frequently I do development on a remote container, and terminal is simply SSHed into that terminal. You could run VS Code in an incredibly lightweight machine, like an Android tablet perhaps, do your actual dev on a container with supercomputer resources, and it'll feel completely natural.

Although I like having a window that appears/disappears when using terminal, sometimes you run a command and a lot of output scrolls by. You wish you had more vertical space. Sure, you can use the "more" command to pause the output, but that's an afterthought. To do so, you can choose to move the terminal to the editor area, as shown in **Figure 7**.

By moving terminal to the editor area, it now becomes a tab, like any other code file within VS Code. Now you can use **CTRL_TAB** to go between your terminal and code or use split view to have your code and terminal side by side. You can even go Zen mode using **CMD_K_Z** to remove all distractions. One trick here is that Zen mode won't get activated with terminal in focus. This is because terminal interprets **CMD_K** differently and essentially cancels out Zen mode. You can open a code file and terminal, focus on the code file, and activate Zen mode. Now you have Zen mode with two tabs: one with code and another with terminal. Neat!

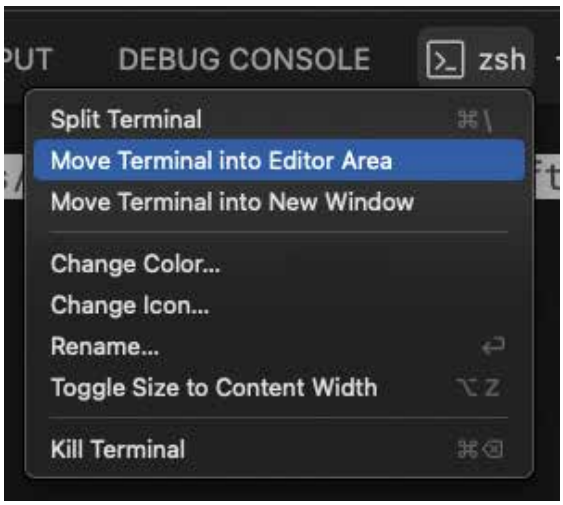

Figure 7: Move terminal to editor area

A long time ago, in a pure Windows world, the only terminal we had was DOS. These days things are a bit more interesting. You have PowerShell, WSL, and so many flavors of Unix shells. VS Code lets you toggle easily between the specific kind of terminal you want. This can be seen in **Figure 8.**

When in terminal, all your usual terminal tricks apply. This isn't an emulated terminal; this is your actual operating system terminal, so all your settings will carry over. All those shortcut keys you are used to, all those nifty aliases and scripts you've set up will still work.

VS Code adds a few more interesting bits on top.

You can use the "code -r" command to open any file within VS Code and reuse the same code window. I find this very useful when I want to view a certain file. I navigate to it using terminal, but I don't want a whole another window to open. Especially on a Mac where ALT_TAB feels wrong, it's so nice to open the file in your current context, grab what you need, press CMD_W to close it, and move on with your life.

The other really nice thing in the integrated terminal is that holding the CMD key down turns a lot of things into links. In fact, VS Code can be taught to customize the behavior of many of these links under terminal settings, but right out of the box, CMD_CLICK on a file opens the file in the current VS Code window.

I confess that I spend most of my time on a Mac and I'm not aware of all the nice things Microsoft has been building into terminal/command prompt on the Windows side lately. But there are a number of things I find useful in the Mac terminal that VS Code makes available inside terminal as hosted by VS Code. I'd be curious to know what your Windows experience is like here. Things like dragging and dropping a file onto terminal and getting the

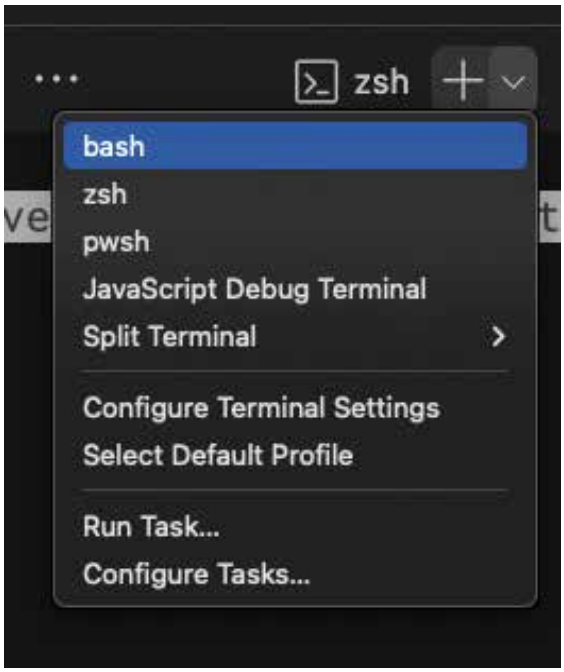

Figure 8: Terminal types

path written out, or pressing CMD_F to search through, or CMD_cursor up or down to iterate between commands, or holding OPTION down and using your mouse to do a vertical select etc.

An interesting default I do like to change, though, is the right-click behavior on the integrated terminal in VS Code. When I right-click in terminal in VS Code, by default, VS Code takes ownership of the right-click behavior and presents a minimum common denominator feature-set. The MacOS terminal offers a number of nifty tricks behind right-click in terminal. For instance, right-clicking on something that looks like a date allows you to create an event or open the calendar for that date. Right-clicking on any word allows you to run the **man** command against that keyword, do automations via services, or search in your favorite search engine.

To allow VS Code to send right-click commands to terminal directly, you can simply set the terminal.integrated.rightClickBehavior setting to "nothing."

What's even more fun is that you can tie all this with profiles. More on that shortly.

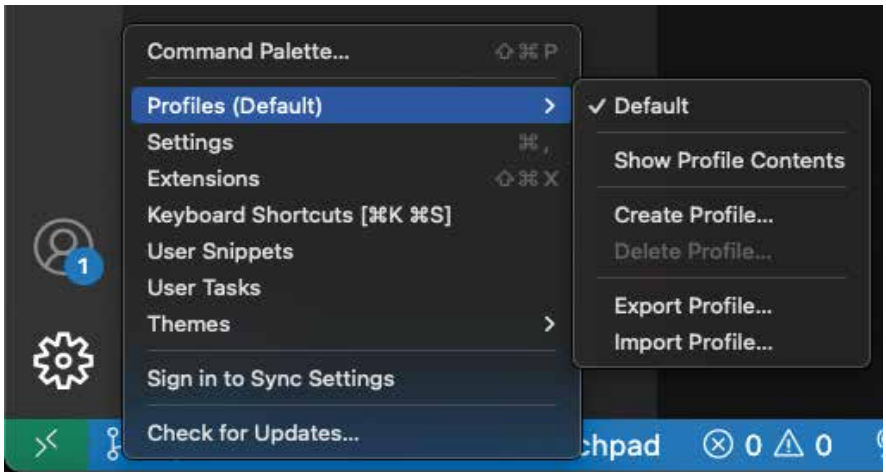

Figure 9: VS Code profiles

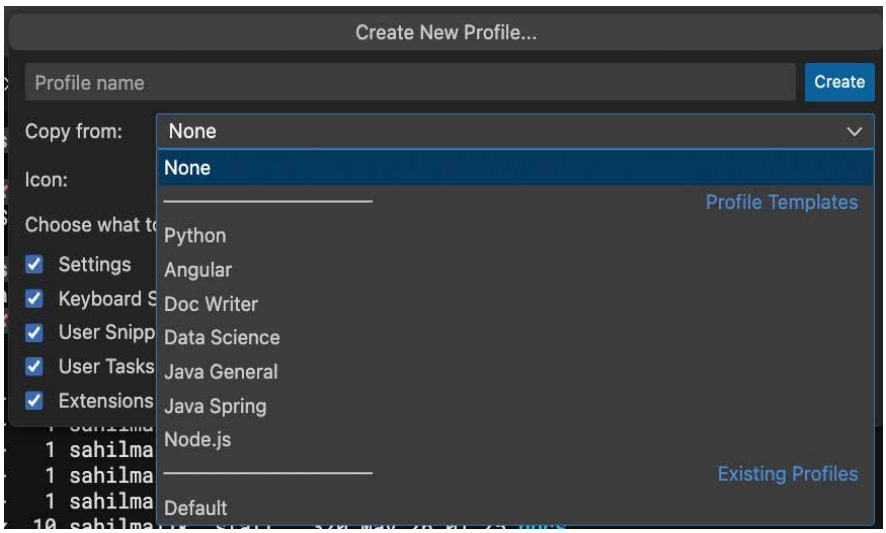

Figure 10: VS Code profile defaults

There are many other interesting tricks you can use with the integrated terminal inside VS Code. For instance, you can right-click on any folder and launch directly into that folder in the integrated terminal, or, via settings, you can configure which terminal you'd rather use. For instance, for my PowerShell projects, I prefer to use **pwsh**. Remember, all these settings are configurable either at the user level or at the workspace level. So if I have a PowerShell project, I want my terminal to be pwsh automatically without having to start terminal and then launch PowerShell.

Profiles

VS Code is an incredibly flexible tool. And it's my and many other developer's de facto tool these days because developers these days are also polyglot. One moment I'm in PowerShell, another in shell scripting, another in JavaScript, then Python, Golang, and sometimes .NET too. The reality is that all these development environments require different settings, preferences, sometimes even colors and fonts.

This is where profiles come in. VS Code profiles let you create sets of customizations that you can use yourself or share with your friends.

By default, all the settings you customize go in the default profile. You can choose to create a new profile or import/export profiles, as can be seen in **Figure 9**.

In fact, when I create a profile, VS Code even helps me pick smart defaults, as can be seen in **Figure 10**.

Because we have a Python project open, go ahead and create a Python profile. I went ahead and created one and called it "Custom Python". Next to the gear icon at the bottom left, VS Code shows me the currently loaded profile. Also, when I choose to edit the profile, it shows

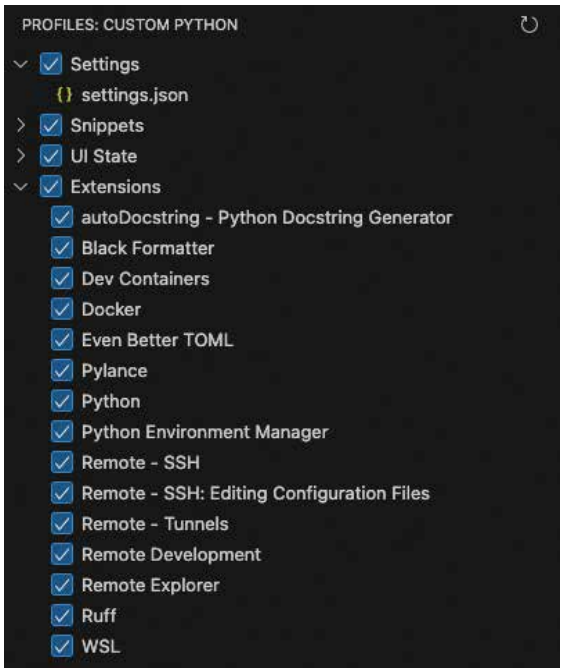

Figure 11: My custom Python profile.

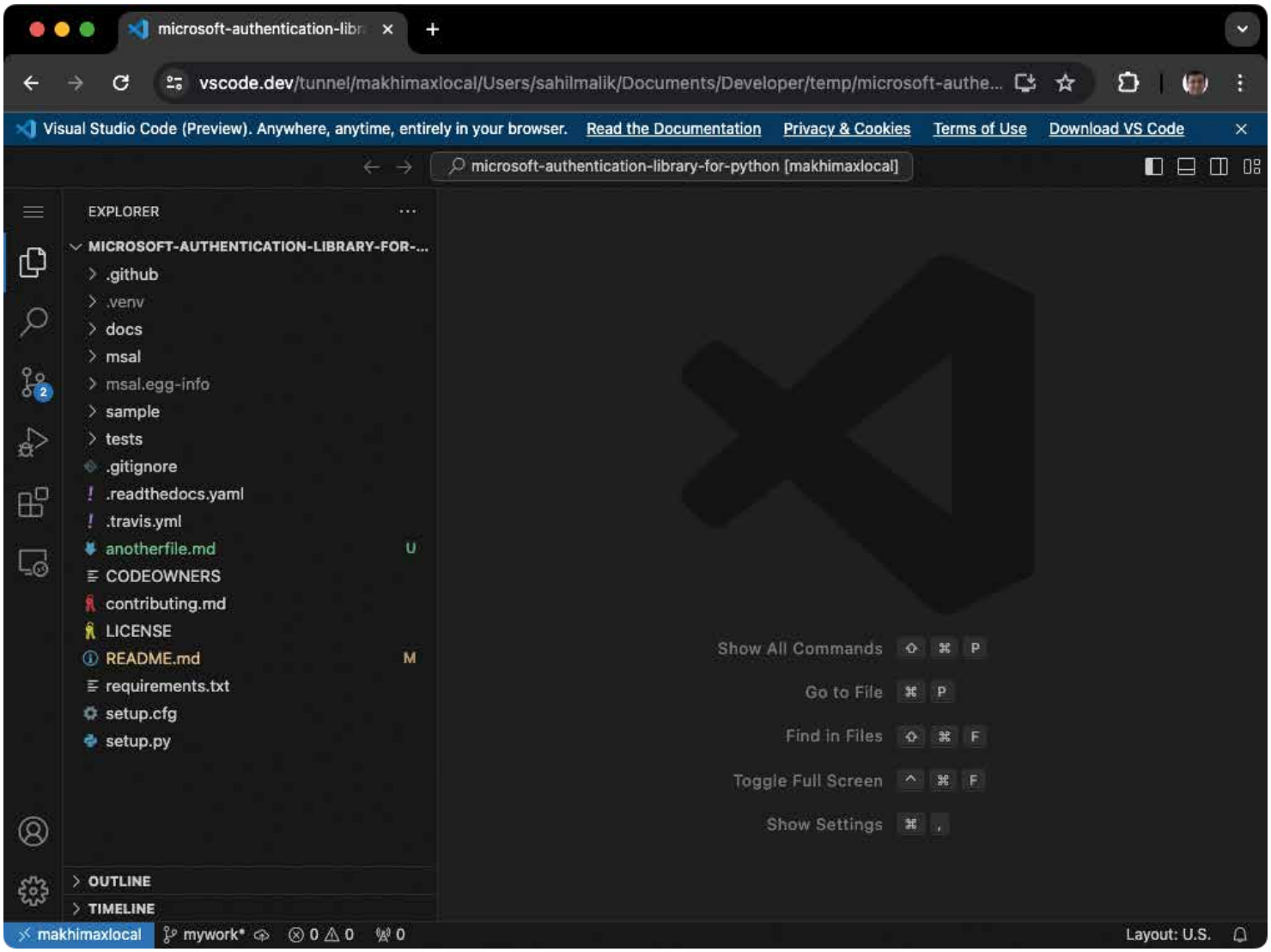

Figure 12: My custom Python profile.

me all the things that are unique to this profile. This can be seen in **Figure 11**.

This is so amazing. Just with a click, I was able to set up my IDE with extensions specific to Python, a dev environment with Docker and Kubernetes support, and even shortcuts and code snippets that are so well suited to Python.

Of course, you'll find things that you want to have available across all profiles. VS Code allows you to apply any setting to all profiles. When you're customizing a setting, just click the gear next to it and choose to apply it to all profiles. Similarly, you can choose to apply an extension across all profiles too.

The other side benefit here, of course, is that when I'm running PowerShell, I don't pay the penalty of loading the Pylance extension. My IDE remains specific and lightweight and suited for the purpose I'm trying to solve. This is very much unlike Visual Studio where the IDE has gotten so bloated over the years that I launch it and go get coffee.

Developing Remotely

This is a technique that I use often and I'm completely blown away by. Sometimes I've been sitting in this chair for 12 hours, and I want to go sit on the couch and work. Or maybe I'm remote desktopped into a Windows machine, and developing on that RDP link is just icky, so I wish I could just develop locally instead but I'm on that remote desktop machine.

This is where the "tunnel" feature comes in handy.

On the remote machine, "cd" in terminal to the folder that contains your workspace and run the following command:

```
code tunnel
```

The first time you run this command, depending upon the auth provider you use, you'll be asked to authenticate and grant permissions. Once you do that, you'll be shown a website URL. This URL is available anywhere in the world and it starts with "vscode.dev".

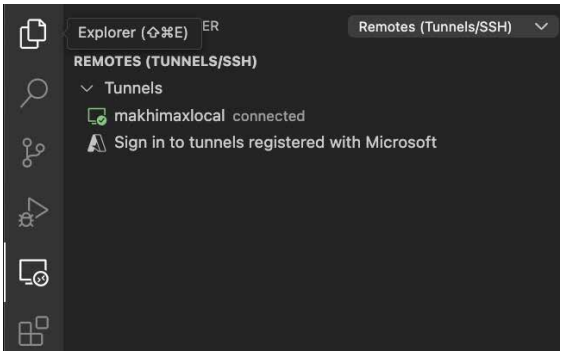

Figure 13: Remote tunnels sidebar

Go ahead and open this URL in a browser on any machine. What do you see? As can be seen in **Figure 12**, this is your project loaded up in the browser, with full VS Code running.

Let's play around with this a bit. Try some shortcut keys like **CTRL_`** to launch terminal, and now you're on command line on your remote machine. How incredible is that?

When someone forces you to develop in an RDP environment, you know what to do, right? Of course, if you don't want to tunnel over the web, you can simply run the following command and get a very similar experience on a local network also.

```
code serve-web
```

Running this command runs a web server on port 8000 by default, which exposes VS Code in a browser that you can access from anywhere on the local network. You can then open any file, folder, or access terminal as you please.

Developing in the browser is cool, but it isn't exactly like developing in VS Code. For instance, the shortcut keys sometimes get confused between the browser and VS Code, and ALT_TAB isn't the same when the Chrome browser is pretending to be VS Code.

Wouldn't it be cool if you could start the VS Code tunnel on a remote machine and then actually use VS Code on a different machine to do your dev work?

You can do so using the remote tunnels extension at https://marketplace.visualstudio.com/items?itemName=ms-vscode.remote-server.

You start the tunnel normally and go to the remote tunnels section on the sidebar, as shown in **Figure 13**.

From this sidebar, you can connect to the tunnel you'd previously created. Once connected, go ahead and open any project on that machine using the File\open folder command. What do you see? Instead of the file picker opening and prompting you to pick a local folder, now you're being asked to open a folder on the remote machine.

Go ahead and open any project. Now you're running a VS Code thick client on a remote machine, and working as if you're on the remote machine. This is 100% seamless and feels totally natural, just like developing on your local machine.

Although this is super cool, what's even more amazing is that you can replicate this on a remote container via the remote-SSH extension. You can effectively do all your dev work on a remote container while VS Code runs on your desktop. It doesn't matter if your desktop is Windows or Mac. The huge win I see here is that all your development is on a remote container that the IT department can secure at their will. This massive dev machine could be provisioned with lots of resources to make you productive. When you're done using the machine, the resources can now be released and shared with another developer in your company.

All this while having zero impact on productivity at your end or subjecting you to an awful development experience.

Try doing this with Visual Studio. This is why VS Code is the IDE that everyone's using.

Summary

I have an unhealthy level of excitement when I talk about VS Code. Having been a developer for a few decades now, I couldn't have imagined that one day we'd have an IDE that's written using JavaScript/TypeScript packaged as an electron app, that runs zippity fast, and that runs everywhere—browser, iPad, Android, Chromebook, Mac, Window. Your code can sit anywhere, you can be somewhere else, it makes IT admins happy, it makes developers happy, and it can support any development environment and programming language.

Name me one other tool that comes even close. A doc writer can use it to take notes, or a Node.js developer can use it to develop websites, or a DevOps developer can use it to orchestrate a complex infrastructure, or a data scientist can use it to sift data, or an AI developer can leverage the power of massive compute on a remote container to fine tune AI models.

I've been lucky enough to have worked personally with the developers who build VS Code. They don't get enough credit. Please give them a shout out at @code.

Until next time, happy coding.

Sahil Malik

Exploring .NET MAUI: Styles, Navigation, and Reusable UI

In Part 1 of this ongoing series on developing an application in .NET MAUI (https://tinyurl.com/3vz3f5j7), you learned the basics of XAML, XML namespaces, attributes, and elements. You created your first .NET MAUI application and ran that application on both a Windows computer and an Android emulator. And you learned how to lay out a basic data-entry page using a Grid and Stack Layouts.

In this article, you'll learn to apply styles so all pages in your application look consistent. You'll create several pages and learn how to use the built-in navigation to move from one page to another. Using a ContentView control, you'll create a header with data bindings that you can reuse on all pages in your application. Finally, you'll add a border and a scroll viewer around all your controls.

Applying Styles to Your Controls

In the XAML you wrote in the previous article Exploring .NET MAUI: Getting Started (https://tinyurl.com/3vz3f5j7), you added attributes to many of the same types of controls to change the **Margin**, **Padding**, and **Spacing** properties. The problem with adding these attributes to individual controls is that if you wish to change them, you must find them all and change them one-by-one. Instead of applying attributes on each control, create a **Style** element to specify which attributes you wish to apply to all controls of the same type. For example, you can create a style that applies to all Label controls or to all Grid controls. Styles avoid you having to set the attributes on individual controls.

There are a few different locations you may place these Style elements. Where you place them has different ramifications as to how those styles are applied to the controls. For example, if you create styles within a ContentPage, those styles are only available on that one page and no others. If you create styles in the App.xaml file, those styles are applied to all pages within the application. You may even create styles in a Class Library that allows many different .NET MAUI applications to reuse those same styles.

Creating Style Elements

Style elements belong inside a <Resources> element. A <Resources> element can be within a single control, on a ContentPage, for the whole application, and within resource dictionary files. The <Style> starting tag must contain a **TargetType** property into which you specify the type of control to target, as shown below.

```
<Style TargetType="HorizontalStackLayout">
  // Add Setter elements here
</Style>
```

Within the Style element, place one or more <Setter> elements. Each Setter element sets a single property to a specific value for the **TargetType** specified. For example, on a HorizontalStackLayout, you can set the Spacing property using the following Setter element. Assign to the **Property** attribute the name of the property on the HorizontalStackLayout you wish to set. Set the **Value** property to the value you want for that property, as shown in the code snippet below.

```
<Style TargetType="HorizontalStackLayout">
  <Setter Property="Spacing"
          Value="5" />
</Style>
```

If you place this style within a <ContentPage.Resources> element at the top of the ContentPage, all the **Spacing** properties on all HorizontalStackLayout controls are set to the value of five (5). If you place this style in the App.xaml file within the <Application.Resources> element, this style applies to all HorizontalStackLayout controls on all pages in the application.

Place Styles on the Content Page

Let's start out by removing all the **Margin**, **Padding**, and **Spacing** properties from the XAML and place those settings into Style elements. Open the **MainPage.xaml** file and just after the starting <ContentPage> element and before the first starting <Grid> tag, add a new element called <ContentPage.Resources>, as shown in **Listing 1**.

Remove the **ColumnSpacing**, **RowSpacing**, **Margin**, and **Padding** from the <Grid> element on this page. Remove the **Spacing="5"** and **Padding="10"** from the <VerticalStackLayout> element on this page. **Remove** the **Spacing="5"** from all <HorizontalStackLayout> elements on this page.

Try It Out

Run the application and notice that the spacing is still the same on the grid and stack layout controls as it was before you removed all the attributes. Also notice that the labels are aligned to each entry control appropriately.

Place Styles at the Application Level

Instead of limiting the defined styles to affect only one page, let's move these styles out of the <ContentPage.Resources> section and move them to the application level so all pages will be styled the same. Open the **MainPage.xaml** file and cut all Style elements from the <ContentPage.Resources> element. Don't remove the <ContentPage.Resources> element from the page yet. Open the **App.xaml** file and paste the <Style> elements you copied **after** the </ResourceDictionary.MergedDictionaries> closing element and **before** the </ResourceDictionary> closing element.

Try It Out

Most often when you change <Style> elements in the App.xaml file, you must restart the application for the changes

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

Paul has been working in the IT industry since 1985. In that time, he has successfully assisted hundreds of companies' 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 multiple courses in the www.pluralsight.com library (https://bit.ly/3gvXgvj) and on YouTube.com (https://www.youtube.com/@pauldsheriff) on topics ranging from C#, LINQ, JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap. Contact Paul at psheriff@pdsa.com.

to take effect. Restart the application and all the spacing should still be applied on your main page.

What's a Resource Dictionary?

In the previous exercise, you placed styles within a <ResourceDictionary> element. A **ResourceDictionary** is a set of resources that can be used in a .NET MAUI application. You may store styles, colors, string and number constants, and data templates within a ResourceDictionary. ResourceDictionary elements can be defined in the App.xaml file as you just saw, or you can create a ResourceDictionary within a separate XAML file, which you'll do later in this article.

Closest Style Wins

Just like CSS in HTML, the closest style to the control it's targeting will take precedence. For example, in the App.xaml file, you defined all Label controls to set their **VerticalOptions** property to "Center". However, open the **MainPage.xaml** file and add the Style within the <ContentPage.Resources> element from the next snippet. This Style element overrides the global style for all labels on this page because it's closer to those controls.

```
<Style TargetType="Label">
  <Setter Property="VerticalOptions"
          Value="End" />
</Style>
```

Try It Out

Run the application and notice that all labels on this page are now positioned toward the bottom of each entry control. After you have observed this change, remove the <ContentPage.Resources> element from the MainPage.xaml so it goes back to using the global styles.

Keyed Styles

All of the styles you created in the App.xaml file are called implicit styles because they implicitly affect all controls set with the **TargetType** property. Another kind of style is called a **Keyed Style**. A keyed style is where you set the **x:Key="UniqueName"** on the starting Style tag. Open the **App.xaml** file and change the <Style TargetType="Grid"> to add an x:Key attribute, as shown below.

```
<Style TargetType="Grid"
       x:Key="Grid.Page">
```

Once you add the x:Key attribute, this style no longer affects all Grid controls. Instead, you need to explicitly reference this key name to apply it to a Grid control.

Try It Out

To see that this style no longer affects a Grid, stop and restart the application to get the global styles to take effect. When the page is displayed, you should notice that the margins are missing on the Grid control.

Apply a Keyed Style

To apply the **Grid.Page** keyed style to a Grid control, add the **Style** property on the Grid you want to apply the style to and reference the name of the key by using the StaticResource markup extension as shown below.

```
<Grid RowDefinitions="Auto,Auto,Auto, ..."
      ColumnDefinitions="Auto,*"
      Style="{StaticResource Grid.Page}">
```

Within the double quotes for the Style property, you're using curly braces and a **StaticResource** markup extension. Think of a markup extension as a .NET class that performs some service for you. In this case, the StaticResource extension searches for a resource defined with the x:Key attribute equal to the value "Grid.Page". When the StaticResource extension locates the resource, it applies any Setter values to the control it's attached to.

Try It Out

Restart the application and you should see that the spacing is once again being applied to the Grid control on your page.

Styling the Shell

To make the title bar area stand out from the rest of the page, add a style to change the background color and the title color. Open the **App.xaml** file and add the following keyed style where you placed the other styles in this file.

```
<Style TargetType="Element"
       x:Key="ShellStyle">
  <Setter Property="Shell.BackgroundColor"
          Value="Blue" />
  <Setter Property="Shell.TitleColor"
          Value="White" />
</Style>
```

Open the **AppShell.xaml** file and in the <Shell> starting tag add the following **Style** attribute to apply the keyed style you just created named ShellStyle.

```
Style="{StaticResource ShellStyle}"
```

Try It Out

Restart the application and you should see that the title bar area is now white text on a blue background.

Place Styles in a Resource Dictionary File

Within your Visual Studio project, drill down into the **Resources\Styles** folder and you should see two files; **Col-**

Listing 1: Add a set of Style elements to apply attributes to the same types of controls.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage ...>

<ContentPage.Resources>
  <Style TargetType="Grid">
    <Setter Property="ColumnSpacing"
            Value="10" />
    <Setter Property="RowSpacing"
            Value="10" />
    <Setter Property="Margin"
            Value="10" />
    <Setter Property="Padding"
            Value="10" />
  </Style>
  <Style TargetType="Label">
    <Setter Property="VerticalOptions"
            Value="Center" />
  </Style>
  <Style TargetType="HorizontalStackLayout">
    <Setter Property="Spacing"
            Value="5" />
  </Style>
</ContentPage.Resources>

<Grid ...>

// REST OF THE XAML
```

ors.xaml and **Styles.xaml**. It's in these two files where the default look is contained for the standard .NET MAUI controls. The Colors.xaml file contains several color resources. The Styles.xaml files uses the colors from the Colors.xaml file to define the look of the .NET MAUI controls as well as setting various properties such as Padding, Heights, Widths, and Fonts of the controls.

Open the **App.xaml** file and within the <ResourceDictionary> element where you placed your styles, there's a <ResourceDictionary.MergedDictionaries> element in which you'll find two ResourceDictionary elements with the **Source** property set to the Colors.xaml and Styles.xaml files respectively. As a normal practice, you'll most likely place your styles after these two files because you'll want to override the default styles.

Previously, you added styles directly in the App.xaml file. You're now going to create your own Resource Dictionary file within the Resources\Styles folder and reference that file by adding your own ResourceDictionary in the <ResourceDictionary.MergedDictionaries> element. Right mouse-click on the **Resources\Styles** folder and select **Add > New Item... > .NET MAUI > .NET MAUI ResourceDictionary (XAML)** from the menu and set the name to **AppStyles.xaml**. Within the <ResourceDictionary> element in this new file, place the XAML shown in **Listing 2**.

Open the **App.xaml** file and just after the ResourceDictionary element that has the **Source** property set to **Resources/Styles/Styles.xaml**, add a reference to your new AppStyles.xaml file, as shown in the following XAML.

```
<ResourceDictionary
   Source="Resources/Styles/AppStyles.xaml" />
```

Because you added all the styles from the App.xaml file into your own resource dictionary file, you may now remove all the styles from the App.xaml file.

Try It Out

Restart the application to ensure that your new AppStyles.xaml file is compiled and your page looks just like it did before.

Override a Style from the Styles.xaml File

To further illustrate that your AppStyles.xaml file overrides the default styles created by Microsoft in the Styles.xaml file, let's change the text and background colors of the Button control. Open the **Resources\Styles\AppStyles.xaml** file and add the following Style element under the **Global Styles** comment.

```
<Style TargetType="Button">
  <Setter Property="TextColor"
          Value="White" />
  <Setter Property="BackgroundColor"
          Value="Black" />
</Style>
```

Try It Out

Restart the application and you should see that the two buttons on your page have white text on a black background. Once you've verified that this works, remove the Button style you just added so it goes back to the default colors.

String Resources

In a ResourceDictionary, you can create other types of resources besides styles. For example, you have repeated the string value "Adventure Works" twice in your application: once in the MainPage.xaml and once in the AppShell.xaml file. Instead of hard coding this string value in multiple places, open the **Resources\Styles\AppStyles.xaml** file and before all other styles, add the following keyed resource.

```
<!-- *************** -->
<!-- Other Resources -->
<!-- *************** -->
<x:String x:Key="ApplicationTitle">
  Adventure Works
</x:String>
```

Now that you have a keyed string resource, reference it from wherever in the application it is needed. Open the **AppShell.xaml** file and modify the **Title** property to reference this keyed resource using the StaticResource markup extension as shown in the following code:

```
Title="{StaticResource ApplicationTitle}"
```

Open the **MainPage.xaml** file and modify the **Title** property to reference this keyed resource using the StaticResource markup extension as shown in the following code:

```
Title="{StaticResource ApplicationTitle}"
```

Try It Out

Run the application to ensure the title appears as you expect on the window shell and your page.

Listing 2: Create a resource dictionary file into which you create your custom styles.

```
<!-- ****************** -->
<!-- ** My Keyed Styles -->
<!-- ****************** -->
<Style TargetType="Grid"
       x:Key="Grid.Page">
  <Setter Property="ColumnSpacing"
          Value="10" />
  <Setter Property="RowSpacing"
          Value="10" />
  <Setter Property="Margin"
          Value="10" />
  <Setter Property="Padding"
          Value="10" />
</Style>
<Style TargetType="Element"
      x:Key="ShellStyles">
  <Setter Property="Shell.BackgroundColor"
          Value="Blue" />
  <Setter Property="Shell.TitleColor"
          Value="White" />
</Style>

<!-- ************* -->
<!-- Global Styles -->
<!-- ************* -->
<Style TargetType="Label">
  <Setter Property="VerticalOptions"
          Value="Center" />
</Style>
<Style TargetType="HorizontalStackLayout">
  <Setter Property="Spacing"
          Value="5" />
</Style>
```

Numeric Resources

In the **AppStyles.xaml** file, there are four **Value** properties in the Grid.Page keyed style set to the number ten (10). Instead of repeating the number ten that many times, create a double resource set to the number ten, as shown in the following XAML:

```
<x:Double x:Key="DefaultSpacingForGrid">
  10
</x:Double>
```

Modify the Grid.Page keyed style to use this new double resource by using the StaticResource markup extension in the Value attribute on each Setter as shown in the following XAML:

```
<Style TargetType="Grid"
       x:Key="Grid.Page">
  <Setter Property="ColumnSpacing"
     Value="{StaticResource
              DefaultSpacingForGrid}" />
  <Setter Property="RowSpacing"
     Value="{StaticResource
              DefaultSpacingForGrid}" />
  <Setter Property="Margin"
     Value="{StaticResource
              DefaultSpacingForGrid}" />
  <Setter Property="Padding"
     Value="{StaticResource
              DefaultSpacingForGrid}" />
</Style>
```

Listing 3: Move resources and styles into the common MAUI library resource dictionary.

```
<!-- *************** -->
<!-- Other Resources -->
<!-- *************** -->
<x:Double x:Key="DefaultSpacingForGrid">
  10
</x:Double>

<!-- ***************** -->
<!-- ** My Keyed Styles -->
<!-- ***************** -->
<Style TargetType="Grid"
       x:Key="Grid.Page">
  <Setter Property="ColumnSpacing"
    Value="{StaticResource
      DefaultSpacingForGrid}" />
  <Setter Property="RowSpacing"
    Value="{StaticResource
      DefaultSpacingForGrid}" />
  <Setter Property="Margin"
    Value="{StaticResource
      DefaultSpacingForGrid}" />
  <Setter Property="Padding"
    Value="{StaticResource
      DefaultSpacingForGrid}" />
</Style>
<Style TargetType="Element"
       x:Key="ShellStyles">
  <Setter Property="Shell.BackgroundColor"
          Value="Blue" />
  <Setter Property="Shell.TitleColor"
          Value="White" />
</Style>

<!-- ************* -->
<!-- Global Styles -->
<!-- ************* -->
<Style TargetType="Label">
  <Setter Property="VerticalOptions"
          Value="Center" />
</Style>
<Style TargetType="HorizontalStackLayout">
  <Setter Property="Spacing"
          Value="5" />
</Style>
```

Try It Out

Restart the application and you should see that the spacing on the Grid is the same as it was before.

Create Styles in a .NET MAUI Class Library

To create a set of styles that you can reuse across multiple .NET MAUI applications, place your styles into a .NET MAUI Class Library project. A class library project is a DLL (assembly) that can be referenced from any .NET MAUI application. Let's add one to the solution by right mouse-clicking on the solution and selecting **Add > New Project...** from the menu. Search for **.NET MAUI Class Library** and select that template.

Set the **Project name** field to **Common.Library.MAUI** and click the Next button. Choose the same .NET version you used to create your .NET MAUI application. Once the application is created, delete the **Class1.cs** file as this isn't needed in this assembly.

I like keeping the same folder structure for my .NET MAUI Class Library projects as for the main application, so right mouse-click on the new project and select **Add > New Folder**. Set the name of this new folder to **Resources**. Right mouse-click on the Resources folder and select **Add > New Folder**. Set the name of this new folder to **Styles**.

Right mouse-click on the Styles folder and select **Add > New Item...** from the menu. Click on the **.NET MAUI** tab and select the template **.NET MAUI ResourceDictionary (XAML)**. Set the **Name** field to **CommonStyles.xaml**. After adding this new ResourceDictionary, add the XAML shown in **Listing 3**. This is the same XAML you have in the AdventureWorks.MAUI project, but you're going to delete that XAML in just a minute.

Go back to the **AdventureWorks.MAUI** project and right mouse-click on the **Dependencies** folder. Choose **Add Project Reference...** from the menu and add a reference to the **Common.Library.MAUI** project you just created. Open the **Resources\Styles\AppStyles.xaml** file in the AdventureWorks.MAUI project and remove everything from this file except the **<x:String x:Key=ApplicationTitle">** element.

Open the **App.xaml** file and add an XML namespace to reference the ResourceDictionary namespace you created in the Common.Library.MAUI project, as shown in the following code. Due to space limitations of this article, I had to break this across multiple lines. Please make sure when adding to your project that the following is all on one line.

```
xmlns:common="clr-namespace:
  Common.Library.MAUI.Resources.Styles;
    assembly=Common.Library.MAUI"
```

The CommonStyles.xaml file you created earlier has a partial class defined in the **CommonStyles.xaml.cs** file.

This class, CommonStyles, is defined within the Common.Library.Resources.Styles namespace. The resources in the CommonStyles.xaml file are compiled into resources that you may now reference from the "common" XML namespace alias you added to App.xaml. Just before the <ResourceDictionary Source="Resources\Styles\AppStyles.xaml" /> element, add the following element.

```
<common:CommonStyles />
```

The styles from the CommonStyles.xaml file are now available for you to use throughout the AdventureWorks.MAUI application.

Try It Out

Run the application and you should see that the main page still looks the same.

Create Several Pages

So far, you've worked only with a single page, MainPage.xaml. In most applications, you're going to have many different pages. You're obviously going to need some way to navigate from one page to another. .NET MAUI has a nice navigation system built-in, but before you can learn about that, you need to create some pages, so let's do that now.

Create a User Detail Page

I prefer to put all my pages underneath a folder named **Views**. Right mouse-click on the project and add a new folder named **Views**. Right mouse-click on the **Views** folder and select **Add > New Item > .NET MAUI > .NET MAUI ContentPage (XAML)...** from the context-sensitive menu. Set the Name of the page to **UserDetailView**. Once the page has been added, change the **Title** attribute on the ContentPage to "**User Information**". Delete the <VerticalStackLayout> element and all XAML within it on this new page. Open the **MainPage.xaml** file, cut the complete <Grid> element from this page and paste it into the ContentPage element on the UserDetailView.xaml file.

Update the Main Page

Open the **MainPage.xaml** and where the <Grid> control was that you cut out, add the <Label> element you see in the following XAML snippet. The MainPage is only going to be used to show the application title in this article series. Other uses for the main page might be as a dashboard with large buttons to navigate to the pages most commonly used, or as description of the application.

```
<Label Text="{StaticResource ApplicationTitle}"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Open the **MainPage.xaml.cs** file and cut the SaveButton_Clicked() event procedure and paste it into the **UserDetailView.xaml.cs** file.

Create Several More Pages

You're now going to create a few more content pages that you can navigate to in your application. On each page, you're going to be adding a Label control with text to identify each page. Right mouse-click on the **Views** folder and add a new **.NET MAUI ContentPage (XAML)** named **LoginView**. Change the **Title** attribute to "Login". Replace the entire <VerticalStackLayout> element with the following XAML:

```
<Label Text="Login"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Right mouse-click on the **Views** folder and add a new **.NET MAUI ContentPage (XAML)** named **ProductDetailView**. Change the **Title** attribute to "Product Information". Replace the entire <VerticalStackLayout> element with the following XAML:

```
<Label Text="Product Information"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Right mouse-click on the **Views** folder and add a new **.NET MAUI ContentPage (XAML)** named **CustomerDetailView**. Change the **Title** attribute to "Customer Information". Replace the entire <VerticalStackLayout> element with the following XAML:

```
<Label Text="Customer Information"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Right mouse-click on the **Views** folder and add a new **.NET MAUI ContentPage (XAML)** named **ColorListView**. Change the **Title** attribute to "Color List". Replace the entire <VerticalStackLayout> element with the following XAML:

```
<Label Text="Color List"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Right mouse-click on the **Views** folder and add a new **.NET MAUI ContentPage (XAML)** named **PhoneTypesListView**. Change the **Title** attribute to "Phone Types List". Replace the entire <VerticalStackLayout> element with the following XAML.

```
<Label Text="Phone Types List"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Navigation in .NET MAUI Applications

Now that you have several pages created, it's time to learn to navigate among the different pages. The Shell control in .NET MAUI uses a URI-based navigation scheme to route between different pages in your application. There are a few different types of navigation schemes that you may employ in your application. You may use a **FlyoutItem**, which is a tab bar for your application and is accessible through an icon, or by swiping from the side of the screen. Another navigation scheme is a **TabBar**, which is a set of tabs displayed either at the top or bottom of the screen depending upon which OS your application is running.

Regardless of which navigation scheme you choose, both **ShellContent** and **Tab** objects are used. A ShellContent object references a single page in your application to which you want to navigate. A Tab object is a wrapper around one or more ShellContent objects. If you're running on Windows, a Tab object with more than one ShellContent object creates a drop-down set of tabs of the enclosed ShellContent objects. If you're running on a mobile device, the Tab object appears as bottom tabs with the ShellContent objects within that Tab shown as top tabs when the bottom tab is selected.

The ShellContent Object

The ShellContent object has many properties, but there are a few that you'll set most often. The **Title** property is used to set the word(s) you want to display on the tab. The **Icon** property is the icon to display along with the text. You also have the **IsChecked**, **IsEnabled**, and **IsVisible** properties to set the currently highlighted tab, whether or not the current tab is enabled, or whether or not it's visible, respectively. The **Route** property is an optional string to uniquely identify the tab/route. If you set the Route property, this property is used to navigate to that specific route. The **ContentTemplate** property is set to a reference of the actual page to display within the application shell. The syntax you use in the ContentTemplate property is a **DataTemplate** markup extension followed by the name of the page. For example, **ContentTemplate="{DataTemplate views:UserDetailView}"** navigates to the UserDetailView page you created.

Using TabBar Navigation

Let's create a TabBar navigation system for the pages you've added to your .NET MAUI application. Because you added all of the pages to the Views folder, they're located within the **AdventureWorks.MAUI.Views** namespace. Thus, if you want to use a page from the Views namespace in a ShellContent object, you need to add a reference to that namespace. Open the **AppShell.xaml** file and add a new XML namespace, as shown in the following code snippet. Please note that I had to break the line just after the colon so it fits within the printed version of this publication. The entire code should be on a single line when you add it to the <Shell> element.

```
xmlns:views="clr-namespace:
  AdventureWorks.MAUI.Views"
```

The <Shell> element currently in the AppShell.xaml file is a single ShellContent object pointing to the MainPage. Delete that ShellContent object from within the <Shell> element and add the following XAML to display tabs to navigate to the home, the UserDetailView, the ProductDetailView, and the CustomerDetailView page:

```
<TabBar>
  <ShellContent Title="Home"
    ContentTemplate="{DataTemplate
      local:MainPage}" />

  <ShellContent Title="Users"
    ContentTemplate="{DataTemplate
      views:UserDetailView}" />

  <ShellContent Title="Products"
    ContentTemplate="{DataTemplate
      views:ProductDetailView}" />

  <ShellContent Title="Customers"
    ContentTemplate="{DataTemplate
      views:CustomerDetailView}" />
</TabBar>
```

Try It Out on Windows

Run the application on **Windows** to display the window shell with the tabs shown in **Figure 1**. Click on each of the tab items to navigate to the corresponding pages. Finally, click on the **Home** tab to navigate back to the Main page.

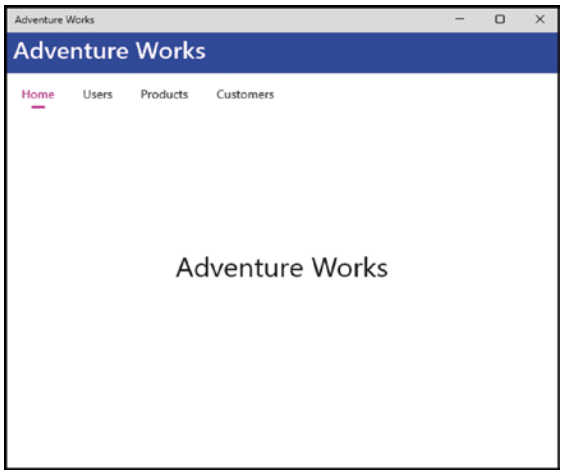

Figure 1: Running the application on Windows displays tab items at the top of the window.

Try It Out on the Android Emulator

Change Visual Studio to run the application on the **Android** Emulator. The tabs should now be displayed at the bottom of the page, as shown in **Figure 2**. Click on each of the various tabs at the bottom of the page to navigate to the corresponding pages. Click on the **Home** menu to navigate back to the Main page.

Display Subordinate Tabs

You can create a tab that doesn't directly navigate to a page, but instead displays a set of tabs either in a drop-down on Windows, or as a set of top tabs on an

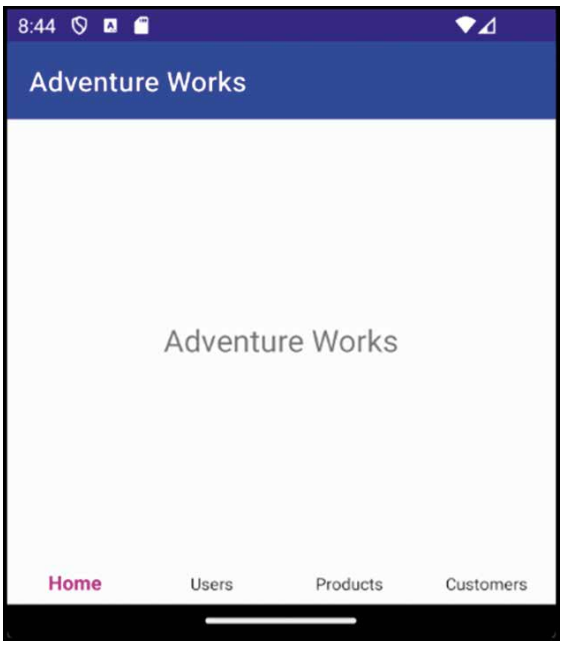

Figure 2: ShellContent objects appear as a set of bottom tabs on an Android device.

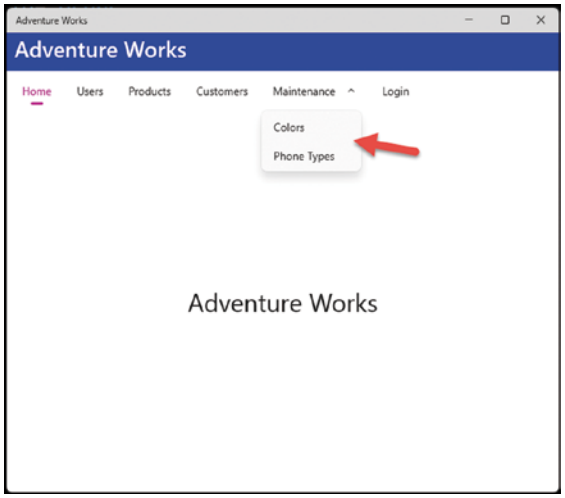

Figure 3: Display a set of drop-down tab items on Windows machines.

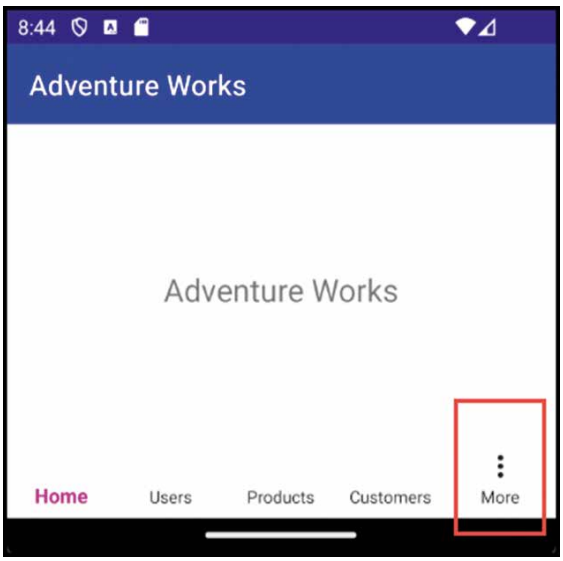

Figure 4: A More tab item is displayed when there are more tabs than fit within the displayable area on a mobile device.

Android device. Open the **AppShell.xaml** file and immediately after the Customers ShellContent object, add the following XAML:

```
<Tab Title="Maintenance">
  <ShellContent Title="Colors"
    ContentTemplate="{DataTemplate
      views:ColorListView}" />

  <ShellContent Title="Phone Types"
    ContentTemplate="{DataTemplate
      views:PhoneTypesListView}" />
</Tab>

<ShellContent Title="Login"
  ContentTemplate="{DataTemplate
    views:LoginView}" />
```

Try It Out on Windows

Restart the application on the **Windows Machine**, click on the down arrow next to the word Maintenance and you should see a drop-down list of tabs, as shown in **Figure 3**.

Try It Out on the Android Emulator

Stop the application and change Visual Studio to use the **Android** Emulator. Run the application and you should now see a **More...** tab displayed, as shown in **Figure 4**. When there are more than four tabs, this **More...** tab is displayed to allow you to get to the next set of tabs.

Click on the **More...** tab to see the **Maintenance** and **Login** tabs appear, as shown in **Figure 5**.

Click on **Maintenance** and you should see **Colors** and **Phone Types** menus appear at the top of the screen, as shown in **Figure 6**.

Programmatically Navigate from Page to Page

Right mouse-click on the Views folder and add a new content page named **UserListView**. Set the **Title** attribute to "User List". Replace the <VerticalStackLayout> with the following XAML:

```
<VerticalStackLayout VerticalOptions="Center"
                     HorizontalOptions="Center"
                     Spacing="10">
  <Label Text="User List"
         FontSize="Header"
         HorizontalOptions="Center" />
  <Button Text="Navigate to Detail"
          Clicked="NavigateToDetail_Clicked" />
</VerticalStackLayout>
```

Create the **Clicked** event procedure by positioning your cursor anywhere within the double quotes of the Clicked attribute. Press the F12 key and the NavigateToDetail_Clicked() event procedure stub is created for you. Modify the procedure declaration to include the **async** keyword and add a single line of code within the procedure, as shown in the following code snippet:

```
private async void
  NavigateToDetail_Clicked(object sender,
    EventArgs e) {
await Shell.Current
    .GoToAsync(nameof(Views.UserDetailView));
}
```

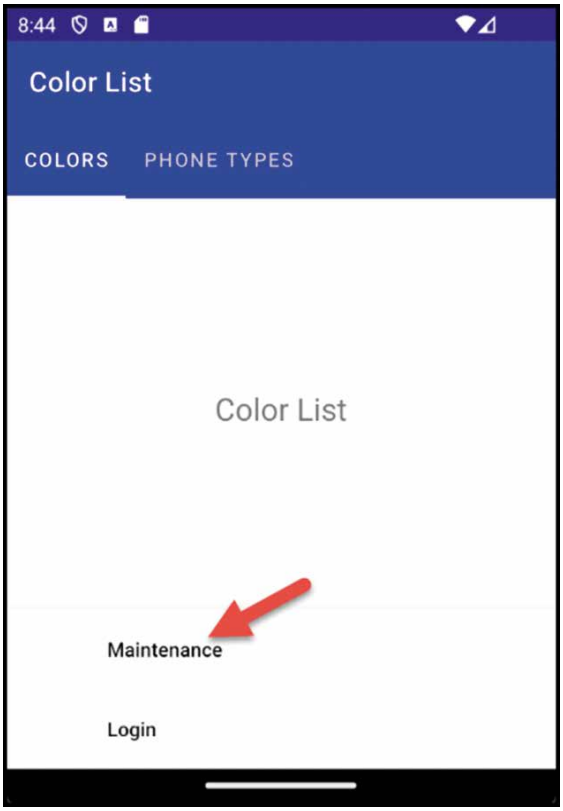

Figure 5: The next set of tabs are displayed after clicking on the More tab.

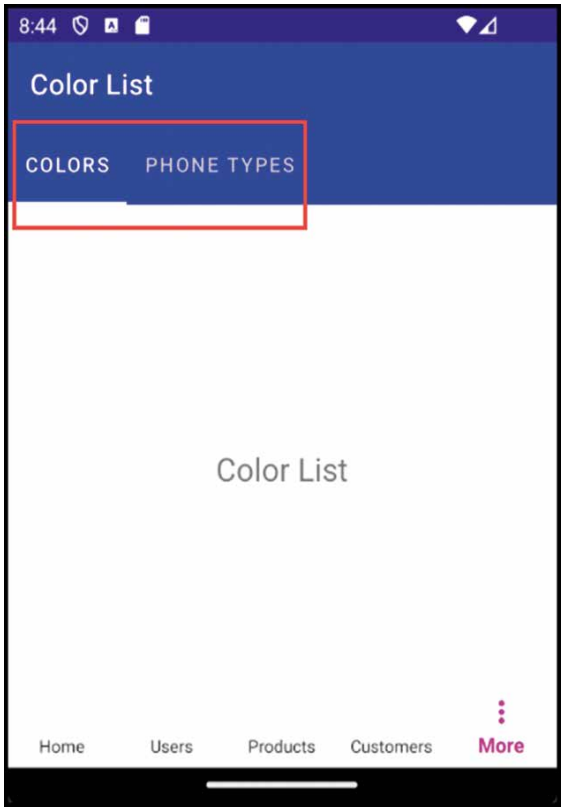

Figure 6: ShellContent objects wrapped within a Tab object are displayed on the top bar on mobile devices.

In the above code, you access the application shell object through the Shell property of the ContentPage class to navigate to another page. Pass the name of the route to the GoToAsync() method and the Shell loads that page and displays it. Always use the C# nameof() function to resolve the route name, as it avoids any typos that could produce a runtime error. If you add the **Route** attribute to a Shell object as a string, you may pass this string to the GoToAsync() method as well.

Modify the Shell to call this new page instead of the user detail page you created earlier. Open the **AppShell.xaml** file and modify the <ShellContent> element that used to call the UserDetailView page and replace it with a call to the UserListView page, as shown in the following XAML:

```
<ShellContent Title="Users"
  ContentTemplate="{DataTemplate
    views:UserListView}" />
```

Register the Route

All of the ShellContent objects in the AppShell.xaml file create a global route table that .NET MAUI uses to navigate to each page. Because you removed the **UserDetailView** ShellContent object, it's now no longer a part of this global route table. If you tried to run the application right now and clicked on the **Navigate to Detail** button, you'd receive a runtime error that the route does not exist. Instead of using XAML to create the route table, you may add as many other routes as you need using C#. Open the **AppShell.xaml.cs** file and in the constructor register the route to the UserDetailView page using the following code:

```
public AppShell() {
  InitializeComponent();

  // Register routes
  Routing.RegisterRoute(
   nameof(Views.UserDetailView),
   typeof(Views.UserDetailView));
}
```

The above code adds the name of the UserDetailView page, and the C# type of UserDetailView to the global route table. Now the routing engine knows which page to display when you request the route name of UserDetailView.

Try It Out

Restart the application and click on the **Users** menu. Click on the **Navigate to Detail** button and you should see the **User Information** page appear.

Create Reusable UI

One of the basic tenets of programming is that you should only write code once, then reuse it as many times as you want. This applies when designing your UI as well. On the user detail view page, the first three rows would be nice to have on all of the pages, so each looks consistent. The first row is a short title to uniquely identify the page. The second row is a short description to describe what this page is used for. The third row is a line to separate the header information from the unique details on the page. Of course, the title and short description should be able to be modified from each page on which these two labels are placed.

To create a custom reusable view onto which you can place any controls and then have those controls appear on any other page, use a **ContentView** control. The ContentView control is a container onto which you place as many controls as you want. You then place the ContentView control onto the page(s) you wish to use that group of controls.

Create a Header View

Let's create a custom "header" view onto which you can place the two labels and the box view control you used on the user detail view page. Right mouse-click on the project and add a new folder named **ViewsPartial**. Right mouse-click on the **ViewsPartial** folder and select **Add > New Item > .NET MAUI > .NET MAUI ContentView (XAML)...** from the context-sensitive menu. Set the name of the ContentView to **HeaderView**. Add to the <ContentView> starting tag an **x:Name** attribute, as shown below, so you may reference the ContentView control named HeaderView by using the name "header".

```
x:Name="header"
```

Replace the <VerticalStackLayout> element of this new file with the XAML shown in **Listing 4**.

The Basics of Data Binding

There are a few pieces of this code that need a little explanation as you're now using data binding. Data binding is a powerful tool that helps you eliminate C# code to control how data is shared between different objects. In the code in **Listing 4**, two Label controls have been placed onto this ContentView control and their **Text** property has been set to a markup extension called **Binding**. This Binding class has a **Path** property to which you set the name of a property on your ContentView control. You're going to create two C# properties in the code-behind of this HeaderView control named **ViewTitle** and **ViewDescription**. There is also a **FallbackValue** property on the Binding class to which you specify a default value if the property being bound to is null or empty.

For the Binding class to know where the property is located, a **BindingContext** must be set either on the Label controls or on one its parent controls. If you look at the

Listing 4: Create a reusable header view that can be placed onto many pages.

```
<Grid BindingContext="{x:Reference header}"
      RowDefinitions="Auto,Auto,Auto">
  <Label Grid.Row="0"
         Text="{Binding Path=ViewTitle,
           FallbackValue='View Title'}"
         HorizontalOptions="Center"
          FontSize="Title" />
  <Label Grid.Row="1"
         Grid.ColumnSpan="2"
         Text="{Binding Path=ViewDescription,
            FallbackValue='View Description'}"
         HorizontalOptions="Center"
         FontSize="Body" />
  <BoxView Grid.Row="2"
           HeightRequest="1"
           Color="Black" />
 </Grid>
```

Listing 5: Connect UI label text with data set into the bindable properties.

```
public string ViewTitle {
  get {
    return (string)GetValue(ViewTitleProperty);
  }
  set {
    SetValue(ViewTitleProperty, value);
  }
}

public static readonly BindableProperty
  ViewTitleProperty =
    BindableProperty.Create("ViewTitle",
      typeof(string), typeof(HeaderView),
      string.Empty);

public string ViewDescription {
  get {
    return (string)GetValue(
      ViewDescriptionProperty);
  }
  set {
    SetValue(ViewDescriptionProperty, value);
  }
}

public static readonly BindableProperty
  ViewDescriptionProperty =
    BindableProperty.Create("ViewDescription",
      typeof(string), typeof(HeaderView),
      string.Empty);
```

Grid control (which is the parent of the Label controls) you see that the **BindingContext** property is set on this control. This **BindingContext** property is set the markup extension **x:Reference**, which references the **x:Name** attribute that has been set on the ContentView control. By referencing the ContentView control using the name, it identifies the HeaderView class is the object that has the properties to reference.

Create Bindable Properties

You now need to create the **ViewTitle** and **ViewDescription** properties on the HeaderView class. You can't just create normal C# properties with simple getters and setters. A **BindableProperty** object is used to create the backing fields for both the properties. By using a BindableProperty object, the UI is automatically notified when a property value changes. Open the **ViewsPartial\HeaderView.xaml.cs** file and add two bindable properties just below the constructor, as shown in **Listing 5**.

Add Header to the User Detail View

Now that you have your reusable header view, it's time to add it to the different pages. Open the **Views\UserDetailView.xaml** file and add the following XML namespace within the <ContentPage>.

```
xmlns:partial="clr-namespace:AdventureWorks
  .MAUI.ViewsPartial"
```

Delete the first two Label controls and the BoxView control within the Grid and replace those controls with the following XAML. You're adding a reference to the HeaderView control located in the namespace aliased by **partial**. Because the **ViewTitle** and **ViewDescription** are public properties, you can set these properties on this control. Be sure that the quoted string for ViewDescription is only on a single line when you type this code into Visual Studio.

```
<partial:HeaderView Grid.Row="0"
  Grid.ColumnSpan="2"
  ViewTitle="User Information"
  ViewDescription="Use this screen to
     modify user information." />
```

You deleted three rows from the Grid control on this page and only replaced one row, so remove two "Auto" values from the **RowDefinitions** property on the Grid. You now also need to renumber the remaining rows in the <Grid> by reducing each by two (2). Go ahead and renumber all the rows now.

Try It Out

Run the application and click on **Users > Navigate to Detail** to view this page with the new reusable header. Notice that the **ViewTitle** and **ViewDescription** properties are set to the values you used in the XAML you just typed in.

Add a Header to the Login View

Open the **Views\LoginView.xaml** file and add the following XML namespace as an attribute on the <ContentPage> element:

```
xmlns:partial="clr-namespace:AdventureWorks
  .MAUI.ViewsPartial"
```

Delete the <Label> element that is currently in this ContentPage and add the XAML shown in **Listing 6** where the <Label> was. Notice that the **ViewTitle** and **ViewDescription** properties are set to different values than those you used on the user detail view page. Also notice the use of the **IsPassword** property on the second Entry control. Setting this property ensures that no one can see the password as you are typing into this control.

Try It Out

Run the application and click on the **Login** menu to view this page. You should see the reusable header above the other controls on this page.

Add a Header to the Product Detail View

Open the **Views\ProductDetailView.xaml** file and add the following XML namespace as an attribute on the <ContentPage> element.

```
xmlns:partial="clr-namespace:AdventureWorks
  .MAUI.ViewsPartial"
```

Delete the <Label> element that is currently in this ContentPage and add the XAML shown in **Listing 7** where the <Label> was.

Try It Out

Run the application and click on the **Products** menu to view this page with the reusable header and the various controls. Don't worry if some of the controls are cropped off at the bottom of the window. You'll fix this later.

Add a Border on All Pages

A **Border** control allows you to draw any size border around a control, or a set of controls. Use the **StrokeThickness** property to set the size in device-independent pixels. The default size is one pixel. The **Stroke** property sets the color for the border outline. The default color is light gray. Open the **MainPage.xaml** file and replace the <Label> element with the following XAML to draw a border around the label on the main page.

```
<Border StrokeThickness="2"
        HorizontalOptions="Center"
        VerticalOptions="Center"
        Padding="10">
  <Label Text="{StaticResource ApplicationTitle}"
         FontSize="Large" />
</Border>
```

Try It Out

Run the application and see the border around the application title on the main page.

Create a Keyed Border Style

I like to wrap a border around each of my pages to make it stand out a little from the edges of the window on which it's hosted. Wrap a Border control around all grids on each page by creating a keyed style to define the standard look for the border on each page. Open the **Resources\Styles\CommonStyles.xaml** file in the **Common.Library.MAUI** project and add the keyed style named **Border.Page** as shown in the XAML below:

```
<Style TargetType="Border"
       x:Key="Border.Page">
  <Setter Property="Stroke"
          Value="Gray" />
  <Setter Property="StrokeThickness"
          Value="1" />
  <Setter Property="Margin"
          Value="10" />
  <Setter Property="HorizontalOptions"
          Value="Fill" />
  <Setter Property="VerticalOptions"
          Value="Fill" />
</Style>
```

Let's now apply this keyed border style on a Border control that you wrap around each Grid control on each of your pages. Open the **Views\LoginView.xaml** file and wrap a <Border> element around the Grid control. Don't forget to add a closing </Border> tag just after the closing </Grid> tag. Add the **Style** attribute to the border and reference the Border.Page keyed style.

```
<Border Style="{StaticResource Border.Page}">
  <Grid ...>
    // REST OF THE XAML HERE
  </Grid>
</Border>
```

Open the **Views\ProductDetail.xaml** file and wrap a <Border> around the Grid control using the keyed style Border.Page.

```
<Border Style="{StaticResource Border.Page}">
  <Grid ...>
    // REST OF THE XAML HERE
  </Grid>
</Border>
```

Open the **Views\UserDetailView.xaml** file and wrap a <Border> around the Grid control using the keyed style Border.Page.

```
<Border Style="{StaticResource Border.Page}">
  <Grid ...>
    // REST OF THE XAML HERE
  </Grid>
</Border>
```

Try It Out

Run the application and click on each menu item of the changed files to view the border around each screen.

Add a ScrollView to all Pages

With the application running, click on the **Products** menu and reduce the main window to show how the controls on the bottom of the page are cropped. Because you never know what the screen size is going to be that the user is running on, it's a good idea to wrap your controls on a page within a scrollable area. This is accomplished by using a **ScrollView** control. Open the **Views\ProductDetailView.xaml** file and wrap a <ScrollView> element around the <Grid> element as shown in the following XAML:

```
<Border ...>
  <ScrollView>
    <Grid ...>
    </Grid>
```

Listing 6: Use the header view partial page on all pages for a consistent look and feel.

```
<Grid RowDefinitions="Auto,Auto,Auto,Auto"
      ColumnDefinitions="Auto,*"
      Style="{StaticResource Grid.Page}">

  <partial:HeaderView Grid.Row="0"
      Grid.ColumnSpan="2"
      ViewTitle="Login"
      ViewDescription="Use this screen to
        login to this application."  />

  <Label Grid.Row="1"
         Text="Login ID" />
  <Entry Grid.Row="1"
         Grid.Column="1"
         Text="" />
  <Label Grid.Row="2"
         Text="Password" />
  <Entry Grid.Row="2"
         Grid.Column="1"
         IsPassword="True" />
  <HorizontalStackLayout Grid.Row="3"
                         Grid.Column="1">
    <Button Text="Login" />
    <Button Text="Cancel" />
  </HorizontalStackLayout>
</Grid>
```

Listing 7 Create the product detail view with many label and entry controls.

```
<Grid RowDefinitions="Auto,Auto,Auto,Auto,
  Auto,Auto,Auto,Auto,Auto,Auto,Auto,
  Auto,Auto,Auto"
      ColumnDefinitions="Auto,*"
      Style="{StaticResource Grid.Page}">

    <partial:HeaderView Grid.Row="0"
      Grid.ColumnSpan="2"
      ViewTitle="Product Information"
      ViewDescription="Use this screen to
         modify product information."  />

    <Label Text="Product Name"
           Grid.Row="1" />
    <Entry Grid.Column="1"
           Grid.Row="1" />
    <Label Text="Product Number"
           Grid.Row="2" />
    <Entry Grid.Row="2"
           Grid.Column="1" />
    <Label Text="Color"
           Grid.Row="3" />
    <Entry Grid.Row="3"
           Grid.Column="1" />
    <Label Text="Cost"
           Grid.Row="4" />
    <Entry Grid.Row="4"
           Grid.Column="1" />
    <Label Text="Price"
           Grid.Row="5" />
    <Entry Grid.Row="5"
           Grid.Column="1" />
    <Label Text="Size"
           Grid.Row="6" />
    <Entry Grid.Row="6"
           Grid.Column="1" />
    <Label Text="Weight"
           Grid.Row="7" />
    <Entry Grid.Row="7"
           Grid.Column="1" />
    <Label Text="Category"
           Grid.Row="8" />
    <Entry Grid.Row="8"
           Grid.Column="1" />
    <Label Text="Model"
           Grid.Row="9" />
    <Entry Grid.Row="9"
           Grid.Column="1" />
    <Label Text="Selling Start Date"
           Grid.Row="10" />
    <Entry Grid.Row="10"
           Grid.Column="1" />
    <Label Text="Selling End Date"
           Grid.Row="11" />
    <Entry Grid.Row="11"
           Grid.Column="1" />
    <Label Text="Discontinued Date"
           Grid.Row="12" />
    <Entry Grid.Row="12"
           Grid.Column="1" />
    <HorizontalStackLayout Grid.Row="13"
                           Grid.Column="1">
      <Button Text="Save" />
      <Button Text="Cancel" />
    </HorizontalStackLayout>
</Grid>
```

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 "Exploring .NET MAUI: Styles, Navigation, and Reusable UI" from the Item drop-down.

```
  </ScrollView>
</Border>
```

Wrap the same <ScrollView> element around the <Grid> element shown in the previous XAML to both the **Views\UserDetailView.xaml** file and the **Views\LoginView.xaml** file.

Try It Out

Run the application, click on the **Products** menu, and reduce the main window size to show that it now displays a scroll bar. Click on the **Users menu > Navigate to Detail button** and see the scroll bar appear on the User Detail page.

Summary

In this article, you used styles to make all pages in your application look consistent. Styles may be placed at many different locations. Which location you choose determines how reusable those styles are. Several pages were created, and you learned how to use the built-in navigation to move from one page to another. Using a ContentView control when you want to create reusable UI for several (or all) pages in your application. Use data binding to change property values on the ContentView control.

Coming up in the next article, you'll start using the many different input controls and learn which ones should be used for specific input data types. In addition, you're going to learn much more about data binding. You'll see how data binding can be used to bind one control to another, and to bind properties in a class to controls on your forms.

Paul D. Sheriff

Job-Oriented Programming and Pointers in a Scripting Language

This article is about using job-oriented programming and pointers in a scripting language. As a scripting language, we'll use CSCS (Customized Scripting in C#). This is an easy and a lightweight open-source language that has been described in previous CODE Magazine articles: https://www.codemag.com/article/1607081 introduced it, https://www.codemag.com/article/1711081 showed how you can use it on top of Xamarin to create cross-platform native mobile apps, and https://www.codemag.com/article/1903081 showed how you can use it for Unity programming. CSCS resembles JavaScript with a few differences, such as the variable and function names being case-insensitive. In this article, we'll be talking about job-oriented programming as implemented in CSCS.

Every office job, like print invoice, create offer, manage customer data, manage product data, print accounting list, etc., has its counterpart in appropriate menu system of business application used. Behind any menu item, there's separate program micro-module. Such micro-modules can interact with each other, calling "child" micro-modules and passing parameters to them. Calling a child micro-module is done by simple **Chain** command, at any line of code. When the "chained" job is done, program execution proceeds with the next line of the parent job (the one that executed the **Chain** command). This way, you can easily develop even huge applications using any number of standalone micro-modules.

You can also send any parameters to the **chained-to** program (the child) and return any data from the child program. This way, you can chain from one program to another, any level deep. Any micro-module can be executed from menu or using Chain command from another micro-module.

This gives developers the freedom to work on a program independently from one another, as a micro-module is self-sufficient regarding the job that it has to do. Passing parameters is just an option.

To facilitate independent development, you need to define parameters that will be used for communication between scripts in case the parameters are needed. This makes any modifications of the whole application very easy, changing only one micro-module.

All this functionality has been made with only two functions: **Chain** and **Param**. The syntax is simple:

```
CHAIN programName WITH arguments;
```

The chained-to program (the child program) can listen for arguments with:

```
PARAM parameters;
```

The arguments can be constants, variables, or expressions, or any combination. Parameters can be a single variable, an array, or a tuple. From a child program, you can access any variable from a previous (parent) program programmatically. You can also return any value from child to parent programmatically.

Let's see how to use CHAIN/PARAM scripting commands in a C# project, taking as an example a WPF (Windows Presentation Foundation) project.

In addition to the CHAIN/PARAM functionality, we'll also look at how to use pointers in a scripting language. It's important to note that usually, the main piece of code is in a compiled language (such as C#), but only a part of the project is done in CSCS scripting. There's an obvious benefit: You don't have to re-compile anything if you change scripting code, because the scripting will be loaded at runtime.

Another benefit of scripting is that you can load scripts, XAML files, and images on the fly from your server. This way, you can get a new version of your client without a formal release! We'll show how to do this in this article.

Let's start by looking at definition of a chain and how chains can be used in scripting.

Setting Up the CSCS Scripting Environment

Probably the simplest way to start using CSCS scripting is to download the source code from GitHub (see https://github.com/vassilych/cscs) and add the source code directly to your C# .NET project. The license lets you modify and use the code without restrictions.

As an example of adding CSCS scripting to a WPF project, take a look at https://github.com/vassilych/cscs_wpf. This is already a preset WPF project with scripting added. There you can also consult the full implementation of the functionality that you'll see in this article.

Take a look at a previous CODE Magazine article: https://www.codemag.com/Article/2008081. It explains how to use WPF with CSCS scripting.

Once you set up the project structure, you can explore the possibilities of the customized scripting that you might not have explored before.

Using Chains in Scripting

In this section, you'll see a cool feature of starting new scripts from a parent script. Also, you'll see how to pass parameters between these scripts.

Nevio Medancic
nevio@aurasoft.hr

Nevio graduated from University of Ljubljana, Slovenia, in IT and Electrical Engineering. Now he's a Senior Software Developer and Product Designer, with a strong foundation in ERP and Accounting Software. He is CEO at Aura Soft in Croatia.

Enzo Grubisa
enzo@aurasoft.hr

Enzo is a C# developer with experience in WPF, Xamarin.Forms, Xamarin.Android, React, and Node.js. He studied computer science at the Faculty of Engineering at the University of Rijeka, Croatia. He currently works at Aura Soft as a main developer.

Vassili Kaplan
VassiliK@gmail.com

Formerly a Microsoft Lync developer, Vassili has also worked on the Microsoft Maquette Mixed Reality Prototyping Tool and adapted the CSCS scripting language to Microsoft Maquette. He has a Masters in Applied Mathematics with Specialization in Computer Science from Purdue University, in Indiana and a Bachelor in Applied Mathematics from ITAM, Mexico City.

Running Child Scripts in the Same Process Space

A sample code for a simple CHAIN functionality is the following:

```
CHAIN "D:/scripts/scriptName.cscs";
```

This runs the specified script and waits until it finishes.

You can also specify the default scripts folder in the standard C# project appSettings configuration file (.config). Here's an example:

```
<add key="ScriptsPath" value="D:/scripts/"/>
```

In this case, you can use the path above with the special tpath() function:

```
CHAIN tpath()+"scriptName.cscs";
```

The **tpath()** is a CSCS function that returns the corresponding "ScriptsPath" path from the config file.

Note that spaces are not allowed inside of an argument (e.g., in tpath()+"scriptName.cscs") because the spaces are the argument separators, as you will see in examples below.

You can also provide parameters for the chained-to script in a simple way:

```
CHAIN "scriptName.cscs" WITH arg1 arg2 arg3;
```

Example:

```
CHAIN tpath()+"scriptName.cscs" WITH "string1"
      variable2 userFunction(arg);
```

Here, **string1** is a string constant, **variable2** can be variable of any type, even an array variable of any size, and **userFunction()** can be any user defined function.

Listing 1: demoMenu.cscs : Main Menu Script

```
// Create main menu window;
// tpath gets the location of the scripts folder
CreateWindow(tpath()+"demoMenuWindow.xaml");

define mainMenuDC type a;

function demoMenuWindow_OnDisplay(){
  AddMenuItem("mainMenuDC", "ScriptsDC", "Scripts");
  AddMenuItem("ScriptsDC", "demoScript1.cscs",
              "Run script1", "runScript");
  AddMenuItem("ScriptsDC", "exit",
              "Exit", "runScript");
}

// function for starting scripts from main menu
function runScript(sender, load){
  if(sender == "exit"){
     // exits the exe program
      exit;
  }
  CHAIN strTrim(tpath())+strTrim(sender);
}
```

Running Child Scripts in a New Process Space

All chained (child) scripts in the previous section run in the same process space. If you want to run the script in another process, there's another syntax of the CHAIN command:

```
CHAIN tpath()+"scriptName.cscs" NEWRUNTIME WITH
      "arg1" "arg2" "arg3" "arg4";
```

In the command above, the NEWRUNTIME keyword tells the interpreter to run the script in another process. Later on, you'll see an example of a script started in a new runtime environment.

Receiving Arguments in the Child Script and Passing Them Back to the Parent Script

Consider a child script, scriptName.cscs, started from the parent's script CHAIN command in the previous section. That example contains four arguments, passed from the parent: strings "arg1", "arg2", "arg3", and "arg4". In the chained (child) script, the arguments from the parent script are received as follows:

```
PARAM(param1, param2, param3, param4);
```

After running the PARAM command above, the chained script sets its local variables as follows:

```
param1 = "arg1"; param2 = "arg2";
param3 = "arg3"; param4 = "arg4";
```

These four variables are local to the chained script and won't be known to the parent.

It's also possible to have a two-way exchange of the parameters between the child and the parent script. The syntax is as follows:

```
ResetField(scriptName, variableName);
```

For example:

```
ResetField("scriptName.cscs", "varNameInParent");
```

After running this command, the chained script has access to the variable defined in the parent script, with the name "varNameInParent". Not only is it going to have access to the variable, but also all of the modifications done to "varNameInParent" are visible in the parent script.

Using the ResetField() command, the child can do both—have access to all of the variables in the parent script, and also to set any variable in the parent—so that after the child is finished, the parent has had all of its variables modified.

Note that the "varNameInParent" variable doesn't have to exist—the child can create it in the parent space and set it to whatever value (including a number, a string, a tuple, etc.) it needs.

Hello, World! Example in Chaining

In this section, you'll see an example of chaining scripts using CSCS scripting language in a WPF C# project. The entry point of the first script to be executed is defined

Listing 2: demoMenuWindow.xaml: Main Menu Window

```
<Window xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/
  presentation"
xmlns:x=
  "http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d=
 "http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc=
"http://schemas.openxmlformats.org/
  markup-compatibility/2006"
xmlns:local="clr-namespace:WpfCSCS"
xmlns:WPF=
"clr-namespace:Microsoft.Web.WebView2.Wpf;
 assembly=Microsoft.Web.WebView2.Wpf"
        mc:Ignorable="d"
        Title="Main menu" Height="450" Width="800"
WindowStartupLocation="CenterScreen">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <WPF:WebView2 Grid.Row="1"
             x:Name="WebView1"
             Margin="0,0,0,0"
     Source= "https://github.com/vassilych/cscs" />
       <Menu Grid.Row="0"
          DataContext="mainMenuDC"
           Name="mainMenuDC" />
  </Grid>
</Window>
```

in the project .exe.config configuration file using the "CSCS_Init" key. Here's what we use in the example:

```
<add key="CSCS_Init"
  value="C:/cscs_wpf/scripts/demoMenu.cscs"/>
```

The full code of the demoMenu.cscs is shown in **Listing 1**.

Note that the first thing the script does is create a new WPF window:

```
CreateWindow(tpath() + "demoMenuWindow.xaml");
```

The full code of the demoMenuWindow.xaml file is shown in **Listing 2**.

As you can see, this is a typical XAML file used in any WPF project. Basically, it just creates a WebView based on the https://github.com/vassilych/cscs website and a simple menu.

This entry window is shown in **Figure 1**:

At the next step, choose "Run script1" menu entry. The demoMenu.cscs initializes the menus as follows:

```
function demoMenuWindow_OnDisplay() {
  AddMenuItem("mainMenuDC", "ScriptsDC",
.             "Scripts");
  AddMenuItem("ScriptsDC", "demoScript1.cscs",
              "Run script1", "runScript");
  AddMenuItem("ScriptsDC", "exit", "Exit",
              "runScript");
}
```

AddMenuItem() is an auxiliary CSCS function that adds an entry to a particular menu.

Here's the CSCS definition of the RunScript function that it uses:

```
function runScript(sender, load) {
  if (sender == "exit") {
     exit;
  }
```

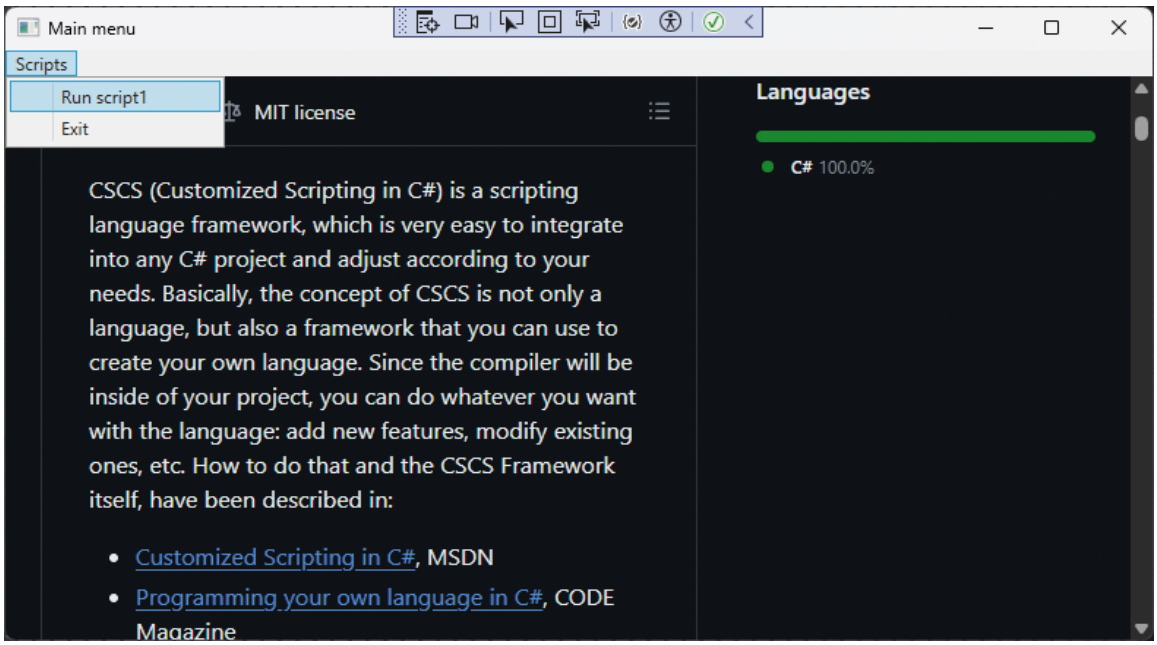

Figure 1: The entry Window after starting a sample project.

Listing 3: demoScript1.cscs: First Script triggered from the Menu

```
CreateWindow(tpath()+"demoScript1.xaml");

function demoScript1_onDisplay(){
    PieChart("dougnutChart1", "init");
    PieChart("dougnutChart1", "values", 50,
             "label 1", 50);
    PieChart("dougnutChart1", "values", 150,
             "label 2", 50);
    PieChart("dougnutChart1", "values", 250,
             "label 3", 50);
    PieChart("dougnutChart1", "values", 350,
             "label 4", 50);
    PieChart("dougnutChart1", "colors",
  { "#1C4E80", "#EA6A47", "#5acc6b", "#34d2fa" });
}

// button clicked event, starts a new script and
// sends parameters("Hello" - constant
// and var1 - variable)
function button1@clicked(){
  var1 = " World!";
  CHAIN strTrim(tpath())+"demoScript2.cscs"
          WITH "Hello" var1;
}

// Same as function above but starts a script in
// a new process
function button2@clicked(){
  var1 = " World!(NEWRUNTIME)";
  CHAIN strTrim(tpath())+"demoScript2.cscs"
        NEWRUNTIME WITH "Hello" var1;
}
```

```
// execute a script from the sender.
  CHAIN tpath()+strTrim(sender);
}
```

Once the user clicks "Run Script1" menu option, the demoScript1.cscs is triggered (see **Listing 3**).

That script creates a new window (see **Listing 4**.). That window is shown in **Figure 2**.

Here we have two buttons; both of them trigger the execution of the demosScript2.cscs script (see **Listing 5**). The left button triggers the execution in the same process and the right one triggers the execution in a new process space.

In any case, as you can see from **Listing 5**, the first thing the script does is to create a new window, demoScript2.xaml (see **Listing 6**).

The new window, created after the chaining, is shown in **Figure 3**.

Both windows (in **Figure 2** and in **Figure 3**) show some graphics. These graphics are parts of the LiveCharts (https://v0.lvcharts.com), an MIT license package that can used as a NuGet package. You can check out how you can create cool graphics from this package in **Listing 3** (Pie Chart) and **Listing 5** (Normal line chart)).

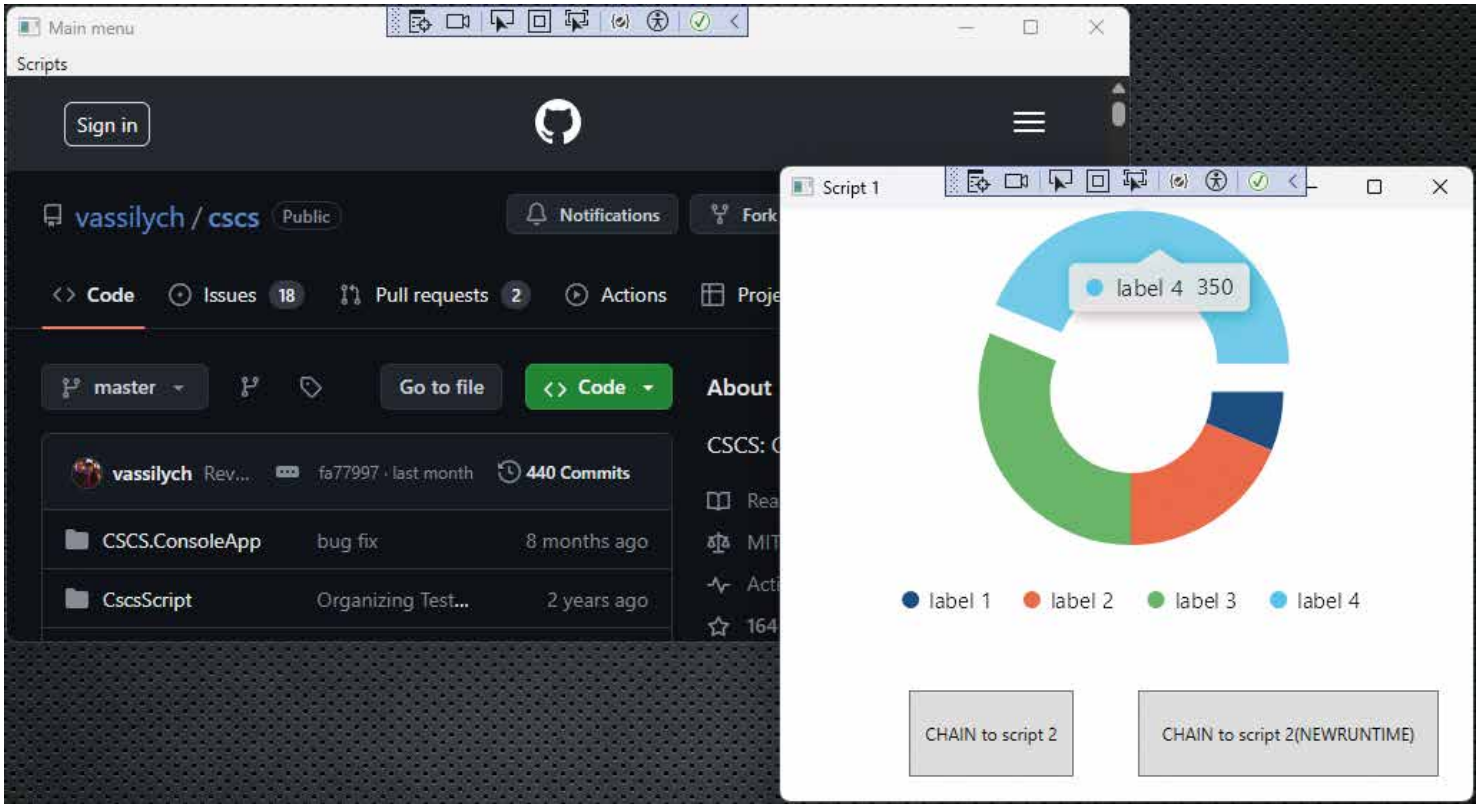

Figure 2: The result of clicking Run Script1 menu

Listing 4: demoScript1.xaml: Window of Script1

```
<Window xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/
  presentation"
  xmlns:x=
  "http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d=
 "http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc=
"http://schemas.openxmlformats.org/
   markup-compatibility/2006"
   xmlns:local="clr-namespace:WpfCSCS"
   xmlns:lvc="clr-
 namespace:LiveChartsCore.SkiaSharpView.WPF;
  assembly=LiveChartsCore.SkiaSharpView.WPF"
  mc:Ignorable="d"
  Title="Script 1" Height="450" Width="500"
    WindowStartupLocation="CenterScreen">
  <Grid>
    <lvc:PieChart LegendPosition="Bottom"
            Name="dougnutChart1"
            VerticalAlignment="Top"
            HorizontalAlignment="Left"
            Width="490"
            Height="289"
            FontSize="10"
            Margin="0,0,0,0" />
    <Button Name="button1"
            Content="CHAIN to script 2"
            HorizontalAlignment="Left"
            Margin="90,334,0,0"
            VerticalAlignment="Top"
            Height="60"
            Width="115" />
    <Button x:Name="button2"
        Content="CHAIN to script 2(NEWRUNTIME)"
        HorizontalAlignment="Left"
        Margin="250,334,0,0"
        VerticalAlignment="Top"
        Height="60"
        Width="210" />
  </Grid>
</Window>
```

Listing 5: demoScript2.cscs: script triggered by demoScript1.cscs

```
CreateWindow(tpath()+"demoScript2.xaml");

DEFINE aPointer type f; // pointer type
DEFINE var1 type i size 10; // integer variable

function demoScript2_onInit() {
  // PARAM will recieve parameters sent from
  // the previous script.
  // They will be named arg1 and arg2 in this script
  PARAM(arg1,arg2);
  // Show arguments
  MessageBox(arg1 + arg2);

  // aPointer points to var1
  aPointer -> var1;

  var1 = 100;
  MessageBox("var1 = " + var1 + ",
              &aPointer = " + &aPointer);
  &aPointer = 200;
  MessageBox("var1 = " + var1 + ",
              &aPointer = " + &aPointer);
}

function demoScript2_onDisplay(){
  Chart("LineChart", "init");
  // x labels first, with font size of 12 px
  Chart("LineChart", "labels","x", 12,
 {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"});
  Chart("LineChart", "SeparatorStep", 1);
  // set of values for the first line chart
  Chart("LineChart", "values", "line",
 {100, 512, 265, 300, 1000, 456, 365, 111, 900, 800},
    "Sample data 1");
  // set of values for the second line chart
  Chart("LineChart", "values", "line",
  {30, 500, 345, 456, 567, 678, 789, 890, 500, 200},
   "Sample data 2");
  // define colors of the first and the second line
  Chart("LineChart", "Color.Series",
        { "#0091D5", "red"});
}

function exitButton@clicked(){
  // exit from the script and return to the
  // previous(parent) script
  quit;
}
```

Downloading Resources from the Internet

In this section, you'll see how to get the resources (all text, .cscs, and .xaml files, and any images) from the internet.

The CSCS function to access a file, located somewhere on the internet, is the following:

> The webserver where you store files for downloading must be configured to treat .cscs files as text files.

```
Download(fullFilename);
```

Note that the webserver where you store your files for downloading must be configured to treat .xaml and .cscs files as text files. Usually, this is already done by default for the .xaml files.

For the .cscs files, change the MIME types on the IIS Manager page, as shown in **Figure 4**.

After setting up your webserver for downloading the .cscs fies, you can access them as follows:

```
newXamlFile = Download( "http://185.32.126.169/
shared/winWithButtons.xaml");
```

You can also do the chaining, explored in the previous section, using scripts from the internet:

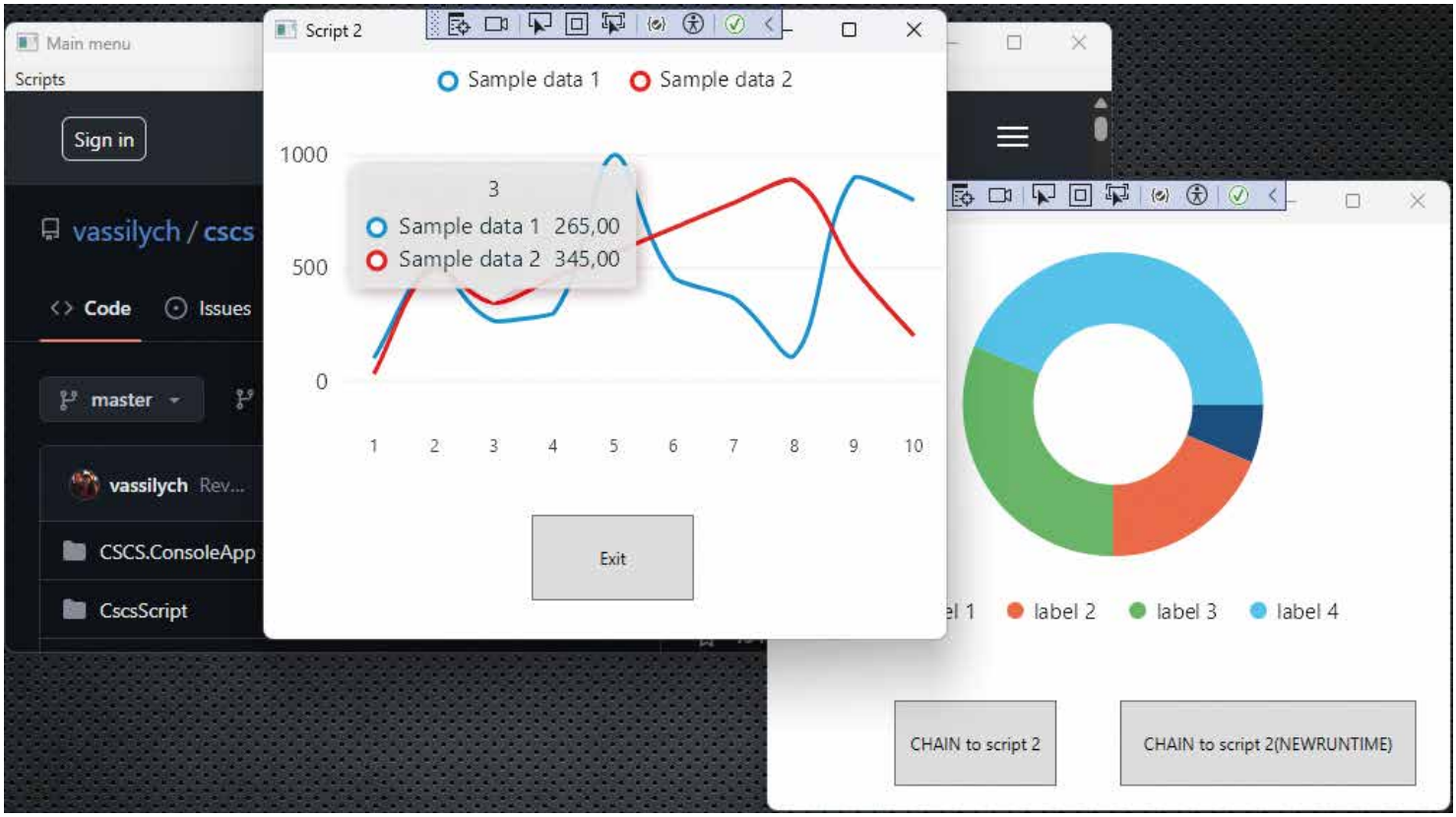

Figure 3: The chained script (the child)

References

CSCS Repository: https://github.com/vassilych/cscs

CSCS WPF Repository: https://github.com/vassilych/cscs_wpf

CSCS Visual Studio Code Debugger Extension: https://marketplace.visualstudio.com/items?itemName=vassilik.cscs-debugger

```
CHAIN Download("http://185.32.126.169/
shared/script5.cscs");
```

This statement means that script5.cscs will be downloaded from an internet location and will be immediately started with just one line of code!

Alternatively, you can download the version of your program and then, if it's not up to date, you can download a new version of the CSCS and /or the XAML files.

Here's a simple pseudo-example how to do it in CSCS:

```
v = Download(
  "http://185.32.126.169/shared/version.txt");
lines = readfile(v);
version = lines[0];
if (myVersion < version) {
  newScriptFile = Download(
   "http://185.32.126.169/shared/script5.cscs");
  copy(newScriptFile, programScriptsDir);
```

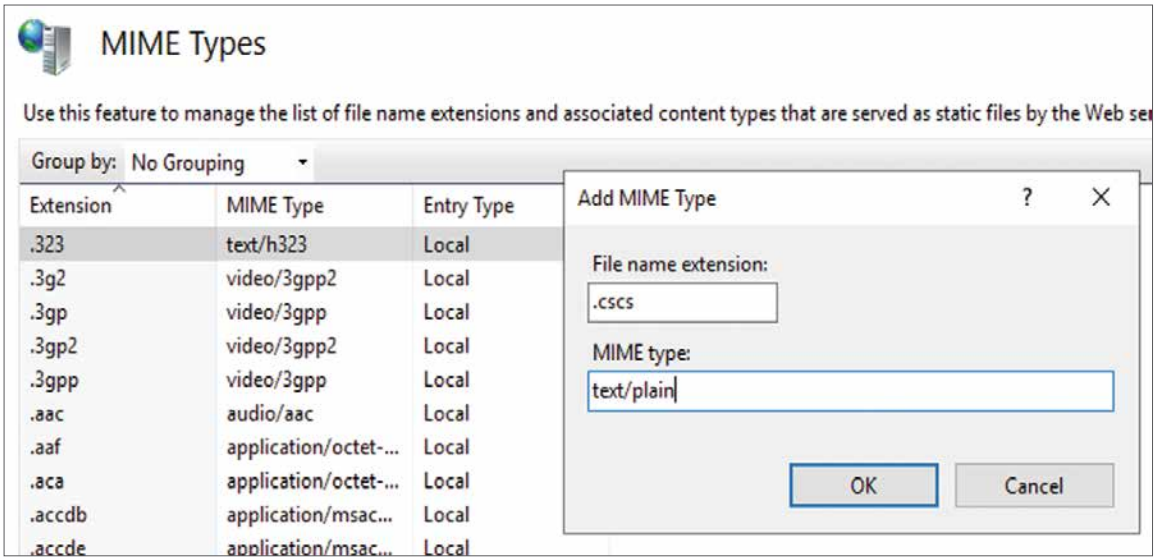

Figure 4: Changing MIME types on the server IIS Manager

Listing 6: demoScript2.xaml: Window of Script2

```
<Window xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/
   presentation"
  xmlns:x=
 "http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d=
 "http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc=
 "http://schemas.openxmlformats.org/markup-
    compatibility/2006"
  xmlns:local="clr-namespace:WpfCSCS"
  xmlns:lvc=
 "clr-namespace:LiveChartsCore.SkiaSharpView.WPF;
  assembly=LiveChartsCore.SkiaSharpView.WPF"
  mc:Ignorable="d"
  Title="Script 2" Height="450" Width="500"
     WindowStartupLocation="CenterScreen">
  <Grid>
    <lvc:CartesianChart LegendPosition="Top"
                        x:Name="LineChart"
                          Margin="0"
                          Height="294"
                          Width="500"
                          HorizontalAlignment="Left"
                          VerticalAlignment="Top" />
    <Button Name="exitButton"
            Content="Exit"
            HorizontalAlignment="Left"
            Margin="190,324,0,0"
            VerticalAlignment="Top"
            Height="60"
            Width="115" />
  </Grid>
</Window>
```

```
  newXamlFile = Download(
    "http://185.32.126.169/shared/win5.xaml");
  copy(newXamlFile, programXamlDir);
}
```

In the same way, you can download images or any other files as well.

Introduction to Pointers in Scripting

Another nice feature that we'd like to introduce in CSCS scripting language is pointers.

Pointers work and look similar to the C/C++ pointers (but not the same though: to dereference a pointer, e.g., to access the value of a variable that it points to, use "&" instead of "*"). An important difference is that you don't have to explicitly allocate and delete the memory to which your pointer points.

After a pointer is defined and initialized to point to a variable, as soon as you change the value of the pointer, the value of the variable to which it points will be changed as well. Also, if you change the value of a variable, the pointer value will also be changed.

Here's an example from **Listing 5**:

```
// A pointer must be first defined with
// a DEFINE keyword
DEFINE aPointer type f;

aPointer -> var1;
// aPointer points now to a variable var1

var1 = 100;
// Now the value of &aPointer will be 100
&aPointer = 200;
// Now the value of var1 will be 200
```

Wrapping Up

In this article, you saw how you can use chains in a scripting language. It's possible to use script chaining to any level, i.e., you can script from the MainMenu script to script1 then from script1 to script2, and so on.

When you quit any script (using the Quit command) the script execution proceeds from the next line after the chain command in the previous (parent) script. Such an implementation can enable even huge applications to be split into many isolated scripts. This is a great advantage for developers who need to independently develop scripts without conflicting with scripts from other developers. As you can see, updating any application may be very easy with simple change of a single script and not of the whole application.

We're looking forward to your feedback, specifically, your experience in scripting with chains and downloading scripts from the internet. Also, we'll be excited to hear what other features in CSCS you would like to see.

Nevio Medancic, Enzo Grubisa, Vassili Kaplan

ANNOUNCING THE 3RD ANNUAL

Power Platform COMMUNITY CONFERENCE

SEPT 18–20 • *Workshops: Sept 16, 17 & 21*

MGM GRAND • *Las Vegas, NV*

CHARLES LAMANNA

Corporate Vice President, Business Industry & Copilot, Microsoft

JEFF TEPER

President – Microsoft Collaborative Apps and Platforms, Microsoft

KIM MANIS

Vice President of Product, Microsoft Fabric & Power BI, Microsoft

SANGYA SINGH

Vice President, Power Platform Intelligent Automations, Microsoft

NIRAV SHAH

Corporate Vice President, Microsoft Dataverse, Microsoft

RYAN CUNNINGHAM

Vice President, Power Platform Applications, Microsoft

ILYA GREBNOV

Vice President, Chief Software Architect, BAP, Microsoft

OMAR AFTAB

Vice President, Conversational AI, Microsoft

REGISTER TODAY!

PowerPlatformConf.com ***@PowerPlatConf*** ***#PPCC24***

Join us at the Grand Garden Arena for our Keynote and Concert!

Copilot • AI • ChatGPT • .NET • ASP.NET • GitHub • Visual Studios • Open AI • LLM • Azure AI Studio • and more...

DEVintersection CONFERENCE & next GenAI CONFERENCE

SEPT 10-12 WORKSHOPS SEPT 8, 9 & 13
MGM GRAND, LAS VEGAS, NEVADA

Learn from Microsoft Leaders, Engineers and Industry Experts:

ERIC BOYD
Corporate Vice President, AI Platform, Microsoft

SCOTT HANSELMAN
Vice President, Developer Community, Microsoft

SCOTT HUNTER
Vice President, Director Product Management for Azure Developer Experience, Microsoft

MADDY MONTAQUILA
Senior Program Manager, .NET MAUI, Microsoft

SAFIA ABDELLA
Senior Software Engineer - .NET, Microsoft

JEFF FRITZ
Principal Program Manager, Microsoft

DAN WAHLIN
Principal Cloud Developer Advocate, Microsoft

JOHN PAPA
Partner GM - Cloud Advocacy, Microsoft

MARKUS EGGER
President and Chief Software Architect, CODE Group

ZOINER TEJADA
CEO and Architect, Solliance

MICHELE LEROUX BUSTAMANTE
President & Architect, Solliance

RICHARD CAMPBELL
Advisor, Creator, Storyteller, Campbell & Associates

REGISTER TODAY!

DEVintersection.com 𝕏@DEVintersection info@DEVintersection.com

nextGenAIconf.com 𝕏@ nextGenAIconf info@nextGenAIconf.com

BONUS: CODE Magazine Track

Questions Answered, Strategies Defined, Relationships Built

CODE Magazine Presents: The State of AI Mini Conference Tour

CODE has recently started a new series of in-person events focusing on the topic of artificial intelligence and its practical uses in business scenarios. We call this our **State of AI: Generating Real Business Value with AI** tour and they're essentially a one-day mini conference with five sessions and two panels. In June 2024 we held events in Houston and Dallas, and by the time you read this we'll present in a few other US cities, with additional locations later in 2024 or early 2025.

You can request an invitation to attend an in-person State of AI event. Because (so far) we are presenting these events in meeting space provided by Microsoft in the local area, we must manage capacity closely. The event features multiple speakers, including CODE Group's founder and Microsoft Regional Director, **Markus Egger, CODE Consulting's CEO, Mike Yeager, CODE Staffing's CEO, Yair Alan Griver, as well as several others from our senior technical staff who are working on AI projects.**

The day's agenda begins with a keynote by Markus Egger, looking at the state of artifi-

r at the Ion District, Houston, TX

cial intelligence and its use in business applications in a way that provides concrete value. (As Markus likes to say: "The time for AI to move from having a lot of potential to providing concrete value is now!") The day then moves on to take a close look at how AI projects fail followed by how to succeed with AI. After a catered lunch, we have a highly anticipated business-focused panel discussion titled **No Answer Begins with "It Depends."** This unique approach has proven to be extremely popular. Audience members can pre-submit questions or ask them on the spot. The panelists are challenged to provide concrete answers, as "It depends" is explicitly not allowed.

> Very insightful presentations by CODE on the practicality of AI in today's business world.
>
> *Chris Zuniga via LinkedIn*

After the business-focused panel and a break, one of CODE's senior software architects presents a deeper technical dive showing how developers might approach using AI in custom applications. The day finishes with a technical panel discussion. We deliberately organize sessions to start less technical in the morning, giving consideration to decision makers who attend, but then end on a more technical note to also provide information to those not afraid of a detailed technical discussion. In our first two events, a large part of the audience stayed beyond the final afternoon break, and the technical panel discussion and Q&A proved to be a very entertaining and informative session.

> As a developer who is exploring the use cases and implementations of AI, this was a very informative and innovative forum.
>
> *Dhinesh Kumar Ramaswami via LinkedIn*

Nevertheless, the overall day is designed to be valuable to non-technical managers and decision-makers, to provide solid information to people who must engage in AI projects or must decide about whether or not such projects even apply to their organization. The day also features breakfast, lunch, and several breaks,

> Being in a room with like-minded AI advocates pursuing the augmentation of human intelligence was truly inspiring… It was fascinating to hear different perspectives on AI's potential.
>
> *Leopoldo Torres via LinkedIn*

> I attended the CODE Magazine State of AI event at the Microsoft Dallas, TX offices today. It was very informative. My biggest take away from this event is ... I can do this.
>
> *Itrel Monroe via LinkedIn*

specifically designed to enable conversations between attendees and CODE speakers and amongst attendees.

Cannot Make It?

For people that cannot make it to an in-person State of AI event, but have an interest in how these topics apply to their company's specific scenarios or businesses, they can request an **AI Executive Briefing**, a service offered by CODE Consulting. AI Executive Briefings are offered in two "sizes." In our (free, online) short version (two hours or so) we provide a presentation to a company's senior managers, and we answer their more urgent questions. In our larger AI Executive Briefing, we have a more in-depth presentation and that leads to an actual pilot project and in-depth technical analysis (individually priced). Either version is a great way to get your management team, executives, or a board of directors up to speed with what AI can provide that is of concrete value for each business. Our AI Executive Briefings are a very popular service, which means we schedule them on a first-come, first-serve basis (and subject to available capacity).

CODE

Container Orchestration Using Kubernetes

In my previous article on Docker ("Introduction to Containerization Using Docker," Mar/Apr 2021 issue), I explained what containerization is and how Docker simplifies application deployment. Docker has revolutionized software development, deployment, and management, especially in the context of microservices. In the realm of microservices, Docker plays a crucial role by providing a consistent and isolated environment for each service. This isolation ensures that services don't interfere with each other, thereby facilitating the development, testing, and deployment of independent application components. Docker's containerization also simplifies scaling microservices, as containers can be easily replicated and orchestrated using tools like Kubernetes.

Wei-Meng Lee
weimenglee@learn2develop.net
http://www.learn2develop.net
@weimenglee

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (www.learn2develop.net), a technology company specializing in hands-on training on the latest technologies. Wei-Meng has many years of training experiences and his training courses place special emphasis on the learning-by-doing approach. His hands-on approach to learning programming makes understanding the subject much easier than reading books, tutorials, and documentation. His name regularly appears in online and print publications such as DevX.com, MobiForge.com, and CODE Magazine.

For large-scale deployments, Kubernetes is often used to manage and scale containers. Although Kubernetes is incredibly powerful and versatile, it's also known for its steep learning curve. Many developers and system administrators find the complexity of Kubernetes daunting because it's challenging to understand and implement effectively. However, tools like minikube offer a simplified way to get started with Kubernetes. Minikube allows you to run a single-node Kubernetes cluster on your local machine, providing a practical environment to learn and experiment without the overhead of a full-fledged cluster.

In this article, I'll guide you through the process of creating a simple Kubernetes cluster using minikube. I'll cover essential Kubernetes components such as Pods, Deployments, Services, NodePort, and ConfigMaps. By the end of this article, you'll have a functional local Kubernetes cluster and a better understanding of how these core elements work together to deploy and manage applications in Kubernetes.

Introducing Minikube

Before deploying applications on a Kubernetes cluster, it's crucial for developers to become familiar with the system. Rather than starting with a live Kubernetes cluster, it's much easier to test on a local test cluster. This is where minikube comes in.

Minikube is a tool that allows developers to run a single-node Kubernetes cluster locally on their personal computers. It's lightweight, easy to install, and effectively simulates how a Kubernetes cluster operates.

Installing Minikube

To install minikube, go to https://minikube.sigs.k8s.io/docs/start/ and select the OS and architecture of the platform you are installing minikube on (see **Figure 1**).

Follow the instructions on the page to begin your installation. For example, if you're using an Apple M-series

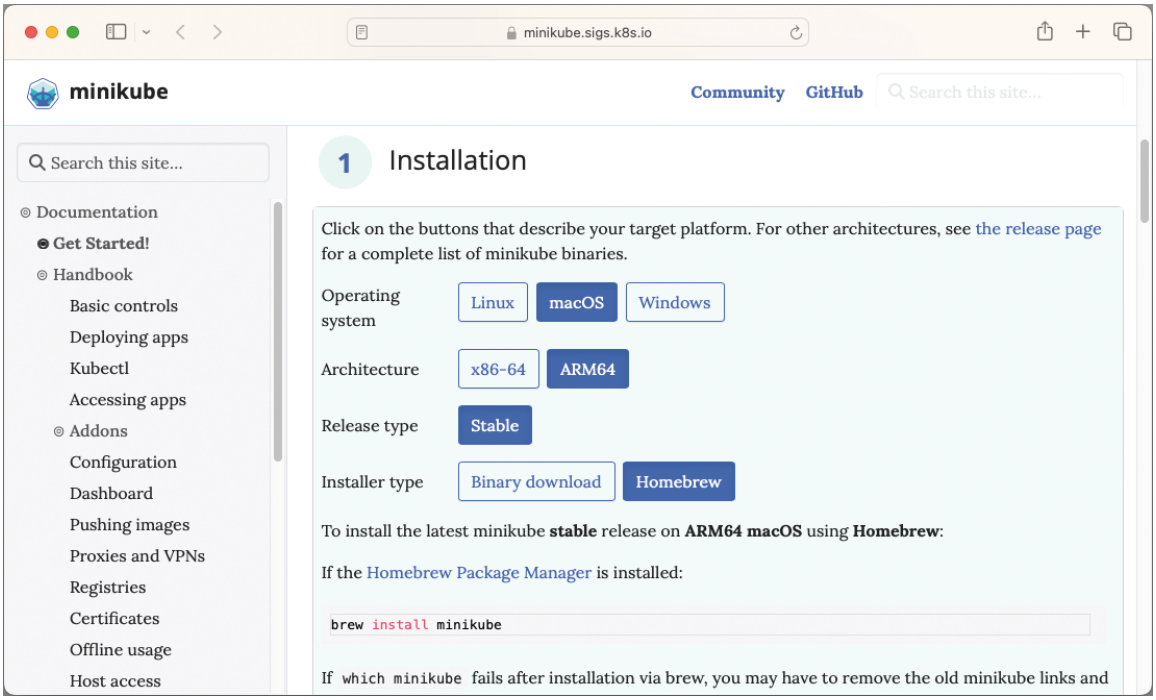

Figure 1: Installing minikube: Choose your OS, architecture, release, and installer type

Mac and have HomeBrew installed, use the following command to install minikube:

```
$ brew install minikube
```

For this article, I'm assuming that you already have Docker installed on your system.

Starting Minikube

Once minikube is installed on your machine, you can simply start it by typing the following command:

```
$ minikube start
```

Figure 2 shows minikube starting up and creating a single-node cluster.

Minikube, by default, uses Docker as the container runtime. You're free to use other container runtimes (e.g., containerd, CRI-O, etc.) with minikube if you wish to. You can check the drivers that minikube is using via the following command:

```
$ minikube profile list
```

Figure 3 shows that minikube is using Docker for both the **VM driver** and the **Runtime**.

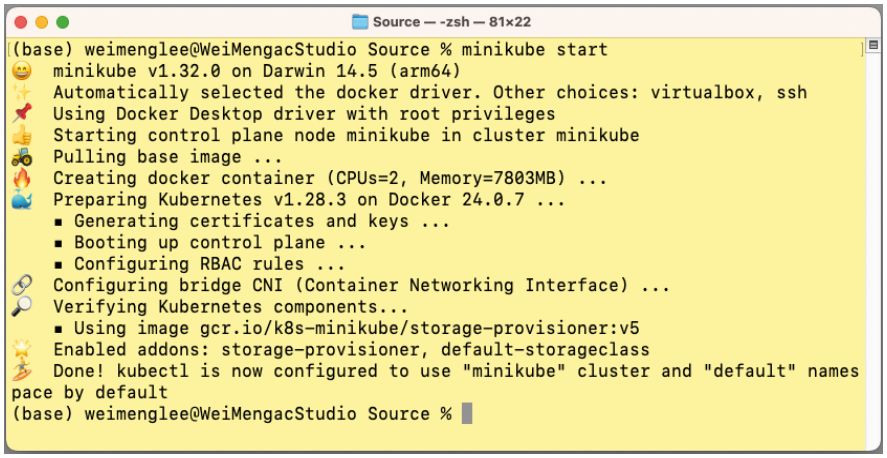

Figure 2: Minikube starting up

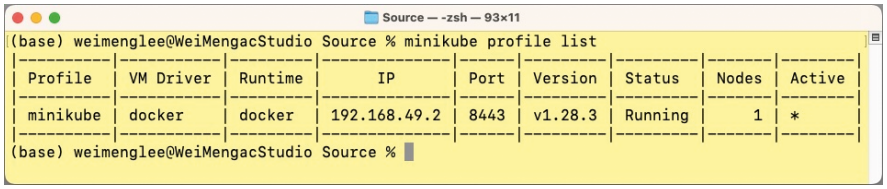

Figure 3: Viewing the drivers that minikube is using

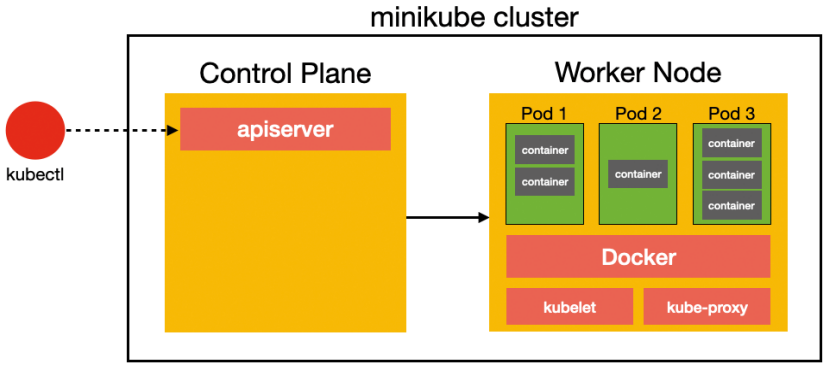

Figure 4: The minikube cluster contains both the control plane and the worker node in a single node (machine).

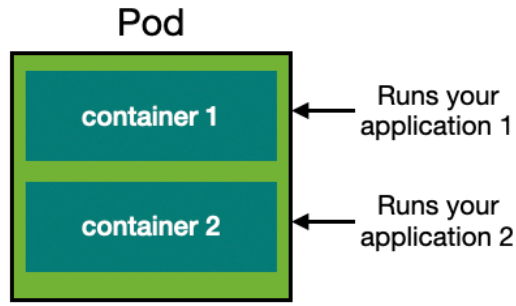

Figure 5: A pod can contain one or more applications.

In this example, using the Docker driver as the VM Driver means that minikube creates Docker containers to manage the various components in your Kubernetes cluster (think of it as minikube using Docker containers to simulate the various nodes in a Kubernetes cluster). Alternatively, you can change the driver to use VirtualBox if you prefer (note that VirtualBox is not supported on Apple's M-series processors):

```
$ minikube start --driver=virtualbox
```

The Runtime, on the other hand, indicates the software minikube uses to manage the containers running inside the Kubernetes cluster. By default, minikube uses Docker as the container runtime. This can be changed. For example, if you want to use **containerd** instead of Docker, you can use the following command:

```
$ minikube start --container-runtime=containerd
```

For this article, I'll be using Docker for both the VM Driver as well as the Runtime for Kubernetes.

To check the status of the minikube cluster, use the following command:

```
$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Here's the breakdown of each of the components in the cluster:

1. **Control Plane:** Manages the Kubernetes cluster. It makes global decisions about the cluster (e.g., scheduling) and detects and responds to cluster events (e.g., starting up new pods when containers fail).
2. **Host:** The container that hosts the Kubernetes cluster.
3. **Kubelet:** The primary node agent that runs on each node. It ensures that containers are running in a pod (more on this later).
4. **Apiserver:** A component of the Kubernetes control plane that exposes the Kubernetes API. It allows clients to connect to the Kubernetes cluster.
5. **Kubeconfig:** A configuration file used by **kubectl** (a client utility) to interact with the cluster. It contains cluster information, user credentials, namespaces, and contexts.

> In real-life, a Kubernetes cluster often has multiple control planes and worker nodes. Each worker node is typically on a separate physical machine or virtual machine. This design allows Kubernetes to distribute workloads across multiple physical or virtual resources, ensuring better resource utilization, fault tolerance, and scalability.

To stop the cluster, use the **stop** option:

```
$ minikube stop
```

To delete the cluster, use the **delete** option:

```
$ minikube delete
```

Using the Various Components in Kubernetes

Now that the minikube cluster is up and running, let's explore the various key components in Kubernetes and learn how they work. You'll progressively build a cluster that contains the following:

1. Pods
2. Deployments
3. Services
4. NodePort
5. ConfigMaps

Pods

A pod is the smallest deployable and manageable unit in Kubernetes. It represents a single instance of a running process in a cluster and can contain one or more containers (usually Docker containers) that share the same network namespace and storage volumes. In short, you run your application within a container hosted in a pod. A pod can contain one or more containers, but it usually contains one.

> A pod represents a single instance of a running process in your cluster, which could be a containerized application, such as Docker container, or multiple tightly coupled containers that need to share resources.

Figure 5 shows a Pod running multiple applications.

Let's now learn how to run an application using a pod. For this example, I'm going to run the **nginx** web server inside a pod. To get Kubernetes to run nginx using a pod, you need to create a YAML configuration file. Let's create a file named **nginxpod.yaml** and populate it with the following statements:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
      - containerPort: 80
    imagePullPolicy: Always
```

Here's what the configuration is doing:

- **apiVersion: v1:** Specifies the API version used to create the pod. For basic pod creation, v1 is commonly used.
- **kind: Pod:** Indicates the type of Kubernetes object being created. In this case, it's a pod.
- **metadata:** Information about information.
  - **name: nginx:** Defines the name of the pod. This name must be unique within the namespace.
  - **labels: name: nginx:** Labels are key-value pairs that are attached to Kubernetes objects. They can be used to organize and select subsets of objects. Here, a label **name: nginx** is added.
- **spec:** Specifies the desired state of the pod.
  - **containers:** A list of containers that will run inside the pod. In this example, there is one container.
  - **name: nginx:** The name of the container within the pod.
  - **image: nginx:** The Docker image (**nginx**) to use for the container. By default, it pulls from Docker Hub.
  - **ports: containerPort: 80:** Exposes port 80 of the container to the network. This is typically where the nginx server listens for HTTP traffic.
  - **imagePullPolicy: Always:** Ensures that Kubernetes always pulls the latest version of the image. This can be useful for development and testing to ensure that the latest image changes are always used.

To apply this configuration to the cluster, use the **kubectl** command:

```
$ kubectl apply -f nginxpod.yaml
pod/nginx created
```

Your pod should now be created. You can verify this by typing the following command:

```
$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          27s
```

To verify that the nginx web server is functioning correctly, you can execute a command in a container running within a pod by using the following command:

```
$ kubectl exec -it nginx -- bash
```

The **exec** command is to execute a command in a container and the **-it** option indicates that you want to run the command in interactive (**-i**) mode with a TTY (terminal, **-t**). The **--** indicates that the following argument (which is **bash**) should be passed to the command executed in the container (and not to **kubectl**).

You should now see the following:

```
root@nginx:/#
```

You are now inside the container running the nginx web server (because the pod only contains one container). To test that the web server works, use the **curl** command as shown in **Listing 1**. If you're able to see the output as shown, your pod running the nginx web server is now working.

Besides going into the nginx container to test the web server, you can also use port forwarding to forward one

Listing 1: Using curl to test the nginx web server

```
root@nginx:/# curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is
 successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please
refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

or more local ports to a port on a pod. The following command forwards the local port 8080 to the port 80 on the pod:

```
$ kubectl port-forward nginx 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

You can now test the nginx web server locally on your machine (open a new Terminal to test this):

```
$ curl localhost:8080
```

It's worth noting at this point that the minikube cluster itself has its own IP address, and so does the pod running within the worker node.

To get the IP address of the minikube cluster, use the following command:

```
$ minikube ip
192.168.49.2
```

To get the IP address of the Pod running within the worker node in the minikube cluster:

```
$ kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP
NODE        NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          15m
10.244.0.4   minikube   <none>
<none>
```

Figure 6 shows the IP addresses of the minikube cluster and the pod.

When you're done with a pod, you can delete it using the **delete** option and specifying the pod name:

```
$ kubectl delete pod nginx
pod "nginx" deleted
```

You can also delete the pod using the configuration file that you used earlier to start the pod:

```
$ kubectl delete -f nginxpod.yaml
```

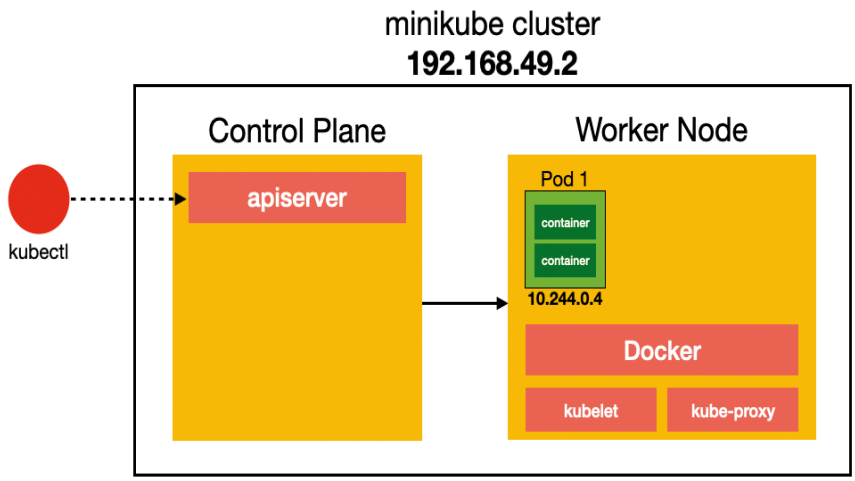

Figure 6: The minikube cluster and the pod each have their own IP addresses.

Deployments

Now that you understand what a pod is, let's move on to the next important component in Kubernetes – **Deployment**. A deployment in Kubernetes is a resource object used to manage the lifecycle of pods. It provides declarative updates to applications, such as rolling out new features or changes, updating existing ones, and rolling back to previous versions if necessary.

> Deployments are a higher-level of abstraction built on top of pods and replica sets, providing more flexibility and ease of management.

In Kubernetes, deployments are often used to manage applications like web servers. Instead of creating a single pod to run nginx, deployments enable the creation of multiple nginx pods. This strategy distributes the work-

Listing 2: Creating a deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-deployment
  labels:
    app: webserver
spec:
  replicas: 5
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "150Mi"
          limits:
            memory: "300Mi"
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 3
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10
```

load across multiple nodes (physical machines), allowing for scalability by dynamically adjusting the number of web server instances in response to changing demand. Additionally, deployments automatically restart failed pods, ensuring high availability of the application.

To see the use of deployments, let's create a new file named **deployment.yaml** and populate it with the following statements, as shown in **Listing 2**.

Here are the key attributes in this configuration file:

- **replicas: 5:** Specifies that you want five replicas (instances) of the pod managed by this Deployment.
- **resources:** Defines resource requests and limits for the container.
  - **requests:** Requests 150MiB of memory.
  - **limits:** Limits memory usage to 300MiB.
- **livenessProbe:** The liveness probe determines if the container is alive and healthy. If it fails, Kubernetes restarts the container.
  - **initialDelaySeconds: 3:** Sets a delay of three seconds before the first liveness probe is performed after the container starts.
  - **periodSeconds: 3:** Sets the frequency at which the liveness probe should be performed, in this case, every three seconds.
- **readinessProbe:** The readiness probe checks if the container is ready to serve traffic. If it fails, Kubernetes removes the container from load balancing until it passes the probe.
  - **initialDelaySeconds: 5:** Sets a delay of five seconds before the first readiness probe is performed after the container starts.
  - **periodSeconds: 10:** Sets the frequency at which the readiness probe should be performed, in this case, every 10 seconds.

This Deployment definition deploys and manages five replicas of the nginx container, ensuring that each pod has defined resource constraints and probes for health checks. To apply this configuration to the cluster, use the following command:

```
$ kubectl apply -f deployment.yaml
deployment.apps/webserver-deployment created
```

You can verify the successful deployment using the following command:

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE
AVAILABLE     AGE
webserver-deployment   5/5     5
5            35s
```

You can see that five out of five pods (5/5 under the READY column) are ready. To view the pods created by this deployment, use the following command:

```
$ kubectl get pods
NAME                                    READY
STATUS    RESTARTS   AGE
webserver-deployment-55f8f4f47c-4lq7r   1/1
Running   0          40s
webserver-deployment-55f8f4f47c-98jvk   1/1
Running   0          40s
webserver-deployment-55f8f4f47c-klntj   1/1
Running   0          40s
webserver-deployment-55f8f4f47c-tqhk8   1/1
Running   0          40s
webserver-deployment-55f8f4f47c-v4kl7   1/1
Running   0          40s
```

From the output, you can see that five pods were indeed created for this deployment. The name of each pod is prefixed with the deployment name, **webserver-deployment**, followed by a unique identifier (**55f8f4f47c** in this case) and additional suffixes to ensure each pod has a unique name (e.g., **4lq7r**).

Each pod has its own assigned IP address. To view the IP address of each pod, use the **-o wide** option:

```
$ kubectl get pods -o wide
```

Figure 7 shows the IP address of each pod.

Figure 8 summarizes the state of the worker node containing the five pods.

Let's try deleting a pod in the deployment using its name:

```
$ kubectl delete pod webserver-deployment-
55f8f4f47c-4lq7r
```

Once that's done, let's check the status of the pod again:

```
$ kubectl get pods
```

You'll observe that a new pod is created to replace the deleted pod (see **Figure 9**). The ability to automatically

```
NAME                                    READY   STATUS    RESTARTS   AGE   IP
webserver-deployment-55f8f4f47c-4lq7r   1/1     Running   0          84s   10.244.0.7
webserver-deployment-55f8f4f47c-98jvk   1/1     Running   0          84s   10.244.0.9
webserver-deployment-55f8f4f47c-klntj   1/1     Running   0          84s   10.244.0.8
webserver-deployment-55f8f4f47c-tqhk8   1/1     Running   0          84s   10.244.0.6
webserver-deployment-55f8f4f47c-v4kl7   1/1     Running   0          84s   10.244.0.5
```

Figure 7: Viewing the IP addresses of each pod

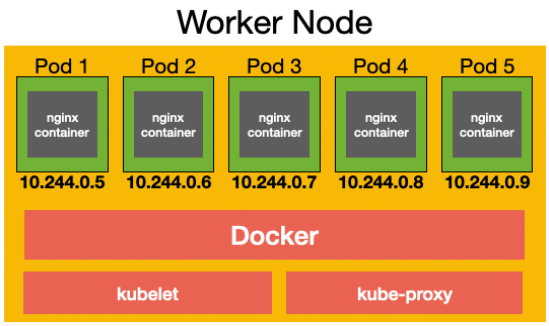

Figure 8: The state of the cluster with the five replicas running nginx

```
NAME                                    READY   STATUS              RESTARTS   AGE
webserver-deployment-55f8f4f47c-98jvk   1/1     Running             0          28m
webserver-deployment-55f8f4f47c-klntj   1/1     Running             0          28m
webserver-deployment-55f8f4f47c-r42kj   0/1     ContainerCreating   0          1s
webserver-deployment-55f8f4f47c-tqhk8   1/1     Running             0          28m
webserver-deployment-55f8f4f47c-v4kl7   1/1     Running             0          28m
```

Figure 9: A new pod is created and started to replace the deleted one

restart a pod that has gone down is one of the advantages of using deployments in Kubernetes.

You can adjust the scale of a deployment as well. For instance, if your deployment currently has five replicas but demand has decreased, you can scale it down to three replicas:

```
$ kubectl scale deployment webserver-
deployment --replicas=3
deployment.apps/webserver-deployment scaled
```

When you now get the pods, you'll see that only three remain:

```
$ kubectl get pods
NAME                                  READY
STATUS    RESTARTS   AGE
webserver-deployment-55f8f4f47c-98jvk   1/1
Running   0          30m
webserver-deployment-55f8f4f47c-klntj   1/1
Running   0          30m
webserver-deployment-55f8f4f47c-v4kl7   1/1
Running   0          30m
```

So now you have three pods running the nginx web server. How do you test them? You can use the port-forwarding technique I discussed in the previous section to port-forward a port on your local server to a particular pod in the worker node:

```
$ kubectl port-forward webserver-deployment-
55f8f4f47c-98jvk 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

In the above command, I forwarded the local port 8080 to the pod named **webserver-deployment-55f8f4f47c-98jvk** listening at port 80. You can now use another terminal to access the nginx web server using the following command:

```
$ curl localhost:8080
```

Services

Up to this point, you've learned that a deployment lets you define how many replicas (or pods) to deploy. When running five pods concurrently, users shouldn't need to interact with a specific pod directly. Instead, users only need to know the website URL, and Kubernetes manages routing requests to the correct pod. This is where **Services** play a crucial role.

A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them. Services enable communication between different parts of the application and can be accessed internally within the cluster or externally depending on the type of service.

Let's now apply a service to the deployment to our nginx deployment so that users don't need to access a specific pod directly. Create a new file named **service.yaml** and populate it with the statements shown in **Listing 3**.

The YAML definition you provided describes a Kubernetes Service named **web-service** that exposes your **webserver** deployment on port 80 using TCP. To apply the configuration to the minikube cluster, use the following command:

```
$ kubectl apply -f service.yaml
service/web-service created
```

You can verify that the **web-service** service is created using the following command:

```
$ kubectl get services
NAME          TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP   10.96.0.1
<none>        443/TCP   26h
web-service   ClusterIP   10.108.174.191
<none>        80/TCP    24s
```

To view more detailed information about the service, use the following command:

```
$ kubectl describe service web-service
Name:              web-service
Namespace:         default
Labels:            svc=web-service
Annotations:       <none>
Selector:          app=webserver
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.108.174.191
IPs:               10.108.174.191
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.244.0.5:80,
```

Listing 3: The service configuration file

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    svc: web-service
spec:
  selector:
    app: webserver
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

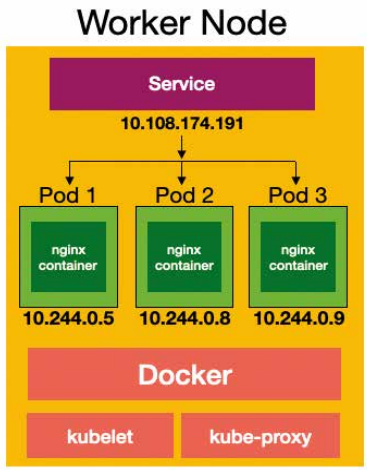

Figure 10: The service has an IP address that all the pods within the node will use to access the nginx web servers.

```
                  10.244.0.8:80,
                  10.244.0.9:80
Session Affinity:  None
Events:            <none>
```

In particular, take note of the following for this example:

- The service has an IP address of **10.108.174.191**. This is the address that all pods within the node will use to access the nginx web servers.
- The **Endpoints** section contains the IP addresses of all the pods that it is connected to. The service will route traffic to the correct pods.

Figure 10 shows the state of the worker node at this moment.

With the service that's created, how do you test the nginx web servers? The first way I want to show you is to create another pod in the node and access the nginx web servers through the service. For this, let's create a new file named **curlpod.yaml** and populate it with the following statements:

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-pod
spec:
  containers:
  - name: curl-container
    image: curlimages/curl:latest
    command:
    - sleep
    - "3600"  # Sleep to keep the
              # container running
```

The above configuration file creates a Pod that contains the curl utility. Using this pod, you can use the curl utility to access the web servers.

Let's apply the configuration to the minikube cluster:

```
$ kubectl apply -f curlpod.yaml
pod/curl-pod created
```

You can now verify that the new pod is created (named **curl-pod**):

```
$ kubectl get pods -o wide
```

Figure 11 shows the curl-pod, along with the three nginx pods.

Figure 12 shows the current state of the worker node.

You can now try to use the **curl-pod** to access the **web-service** service:

```
$ kubectl exec -it curl-pod  -- curl
web-service
```

The above command uses the **curl-pod** to issue a curl request to the nginx web server through the **web-service** service.

```
NAME                                   READY   STATUS    RESTARTS   AGE   IP
curl-pod                               1/1     Running   0          53s   10.244.0.12
webserver-deployment-55f8f4f47c-98jvk  1/1     Running   0          23h   10.244.0.9
webserver-deployment-55f8f4f47c-klntj  1/1     Running   0          23h   10.244.0.8
webserver-deployment-55f8f4f47c-v4kl7  1/1     Running   0          23h   10.244.0.5
```

Figure 11: The curl-pod along with the three nginx pods

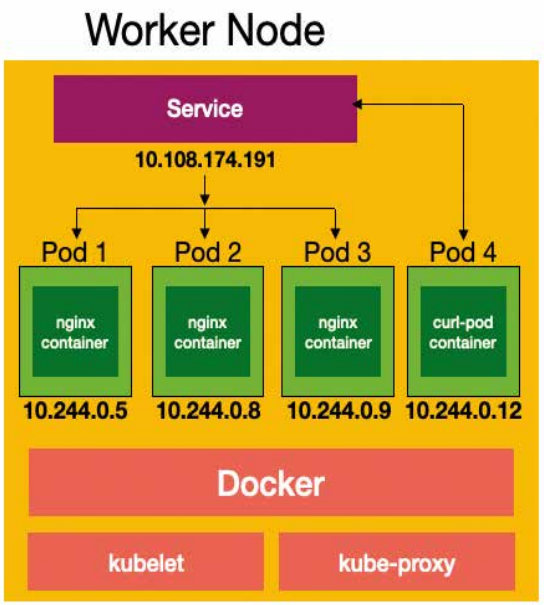

Figure 12: The addition of the curl-pod to the worker node

Alternatively, you can also use the IP address of the **web-service** service:

```
$ kubectl exec -it curl-pod  -- curl 10.108.174.191
```

If you wish to access the web-service service from your local machine, you can use port forwarding. The following command is used to create a port-forwarding tunnel from your local machine to the **web-service** service:

```
$ kubectl port-forward service/web-service
8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

You can now use another terminal to access the nginx web server using the following command:

```
$ curl localhost:8080
```

NodePort

A Service in Kubernetes allows pods in a cluster to communicate with each other, regardless of whether they're running on the same node or different nodes. What happens if you want to expose the service so that it's accessible from outside the cluster? This is where **NodePort** comes into the picture. NodePort is a specific type of Service that exposes the Service on a static port (the NodePort) on each Node's IP address. When you create a NodePort Service, a **ClusterIP Service**, to which the NodePort Service routes, is automatically created. The NodePort Service makes it possible to access the application using **<NodeIP>:<NodePort>** from outside the cluster.

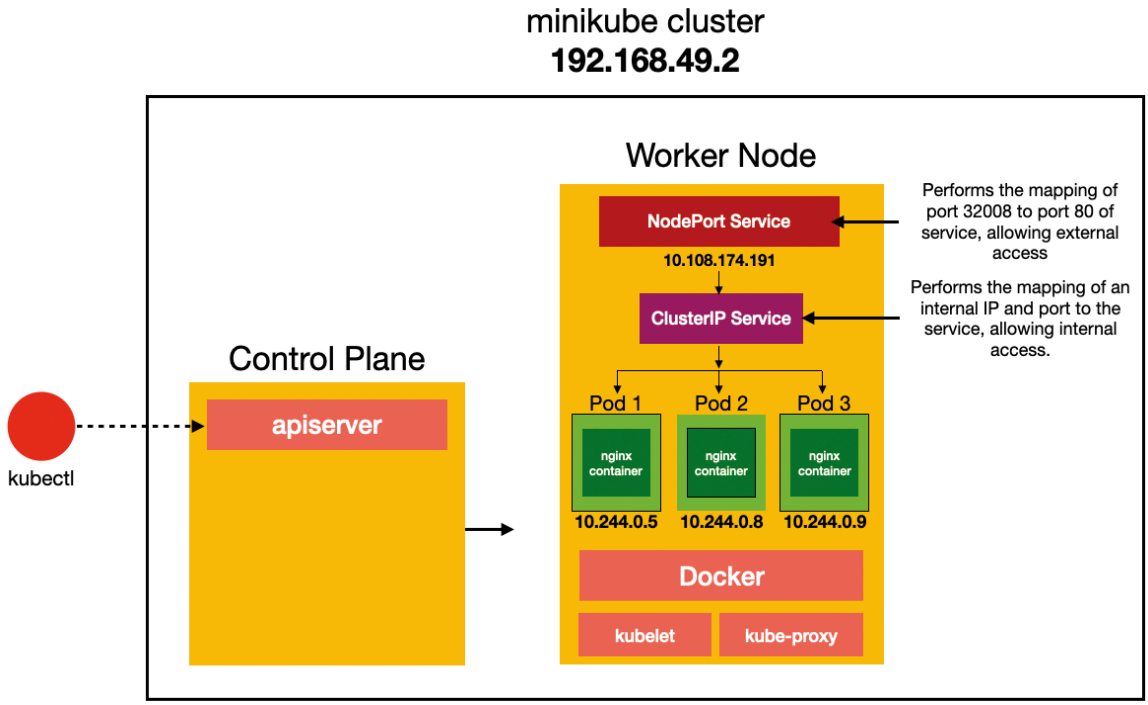

Figure 13: The cluster with the NodePort listening at port 32008

```
|-----------|-------------|-------------|---------------------------|
| NAMESPACE |     NAME    | TARGET PORT |            URL            |
|-----------|-------------|-------------|---------------------------|
| default   | web-service |          80 | http://192.168.49.2:32008 |
|-----------|-------------|-------------|---------------------------|
🏃  Starting tunnel for service web-service.
|-----------|-------------|-------------|------------------------|
| NAMESPACE |     NAME    | TARGET PORT |          URL           |
|-----------|-------------|-------------|------------------------|
| default   | web-service |             | http://127.0.0.1:56554 |
|-----------|-------------|-------------|------------------------|
🎉  Opening service default/web-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Figure 14: Creating a tunnel to the service

Let's see this in action. First, create a file named **service_nodeport.yaml** and populate it with the following statements:

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    svc: web-service
spec:
  type: NodePort
  selector:
    app: webserver
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 32008
```

This configuration defines a NodePort service. Similar to a regular service, it includes an additional attribute called **nodePort,** which specifies the port (e.g., 32008) on each node where the service will be accessible. To apply this configuration, use the following command, as usual:

```
$ kubectl apply -f service_nodeport.yaml
service/web-service configured
```

You can now verify that the NodePort is created correctly:

```
$ kubectl get svc web-service
NAME          TYPE       CLUSTER-IP
EXTERNAL-IP   PORT(S)        AGE
web-service   NodePort   10.108.174.191
<none>        80:32008/TCP   9h
```

Figure 13 shows the current state of the cluster.

You should now be able to access the nginx web server through the node's IP address which, in the case of minikube, is also the minikube IP address. However, in the case of minikube, you have to create a tunnel to the service via ports published by the minikube container, which will then perform a port forwarding for you:

```
$ minikube service web-service
```

Figure 14 shows that result returned by the above command.

Figure 15 shows that to access the node's web server, you have to use the address **127.0.0.1:56554**, which will then redirect the traffic to the minikube cluster's IP address (port 32008 as specified by the node port), which will then direct to the **ClusterIP Service.**

Figure 16 shows the page returned by the nginx web server.

Listing 4: The content of the ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: index-html-configmap
data:
  index.html: |
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport"
        content="width=device-width,
                 initial-scale=1.0">
      <title>Hello Kubernetes</title>
      <style>
        .tray {
          display: inline-block;
          background-color: LightCoral;
          color: white;
          padding: 10px;
          border-radius: 5px;
        }
      </style>
    </head>
    <body>
      <center>
        <h1 class="tray">Hello, Kubernetes!</h1>
      </center>
    </body>
    </html>
```

ConfigMaps

You now have the nginx web servers ready to serve from within a Kubernetes cluster, with built-in redundancies. The web servers are accessible through a NodePort Service, which redirects traffic from outside the cluster to the pods within the cluster. But how do you change the content that nginx is serving? One way is to use a **ConfigMap**.

A ConfigMap in Kubernetes is an API object used to store non-confidential configuration data in key-value pairs. ConfigMaps allow you to decouple configuration artifacts from container images, enabling you to keep application configuration separate from your containerized application code. They're often used to store configuration data such as application settings, environment variables, command-line arguments, and configuration files. They're commonly used when deploying your database servers in Kubernetes.

For this example, I'll use ConfigMap to set the **index.html** page that nginx will serve. Let's create a new file named **configmap.yaml** and populate it with the statements shown in **Listing 4**.

This ConfigMap (named **index-html-configmap**) contains an **index.html** file with the specified HTML content. You'll configure your nginx web server to serve this file (**index.html**) when it's accessed.

You need to apply this ConfigMap to the minikube cluster using the following command:

```
$ kubectl apply -f configmap.yaml
configmap/index-html-configmap created
```

Next, modify the **deployment.yaml** file by appending the statements as shown in **Listing 5**.

In the above, the deployment configuration defines a volume named **nginx-index-file** that obtains its data from the **index-html-configmap** ConfigMap (defined in the **configmap.yaml** file). The **volumeMounts**: section mounts the **nginx-index-file** volume at **/usr/share/nginx/html/index.html**.

Figure 17 shows the relationship between the ConfigMap and the Deployment.

Apply the configuration to the cluster using the following command:

```
$ kubectl apply -f deployment.yaml
deployment.apps/webserver-deployment created
```

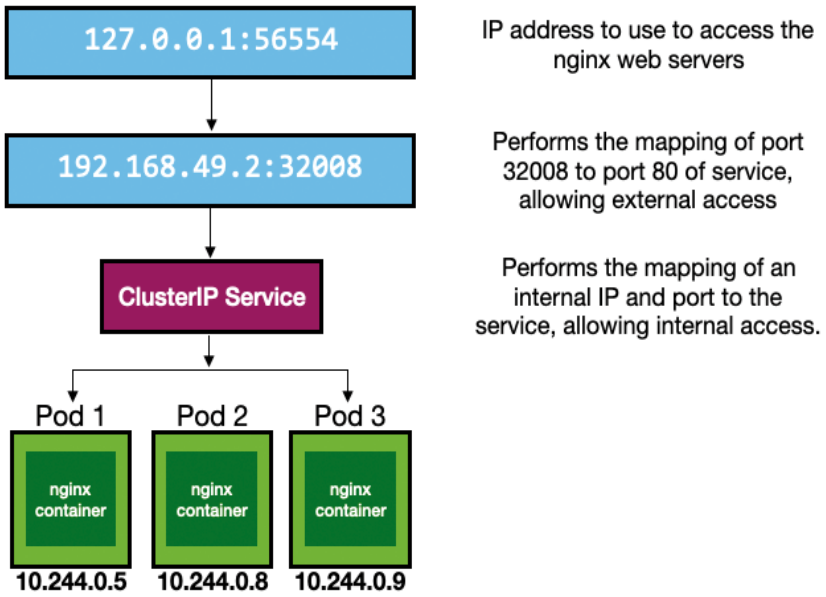

Figure 15: Accessing the nginx web servers using the local IP address

Figure 16: The web page returned by the nginx web server

Listing 5: Adding the volumes: and volumeMounts: to the deployment file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-deployment
  labels:
    app: webserver
spec:
  replicas: 5
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "150Mi"
          limits:
            memory: "300Mi"
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 3
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10
        # add the following statements
        volumeMounts:
        - name: nginx-index-file
          mountPath: "/usr/share/nginx/html/"
      volumes:
      - name: nginx-index-file
        configMap:
          name: index-html-configmap
```

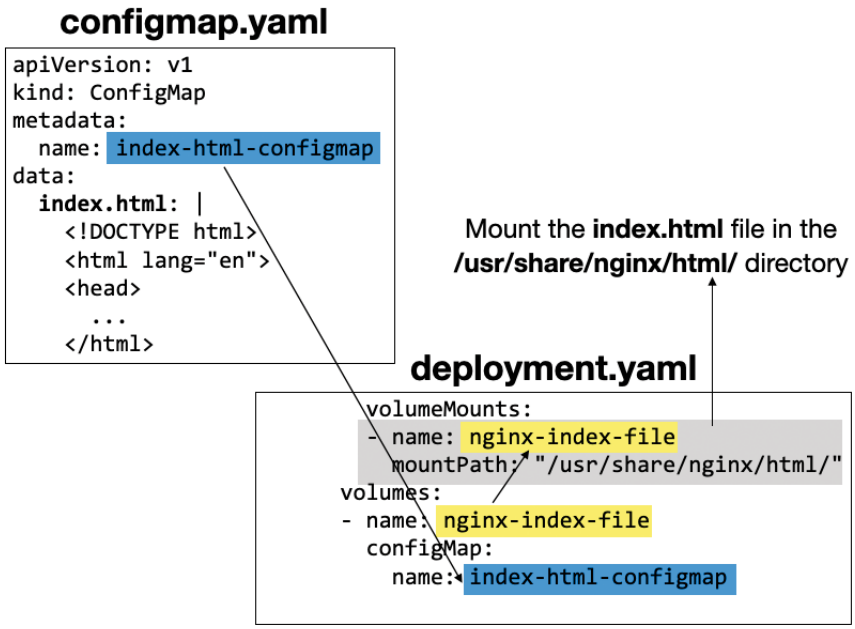

Figure 17: The relationship between the ConfigMap and Deployment

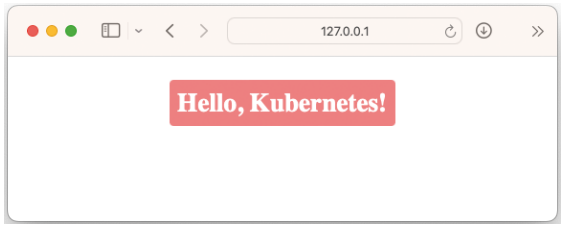

Figure 18: The nginx web servers now serving the index.html file that you've created in the ConfigMap

You can now create a tunnel to the service via ports published by the minikube container, which will then perform a port forwarding for you:

```
$ minikube service web-service
```

Figure 18 shows the page now served by the nginx web servers.

Summary

In this article, I explored the foundational concepts of Kubernetes, focusing on its role in modern application deployment and management.

To ease the learning curve associated with Kubernetes, you can use minikube, a tool that simplifies setting up and running Kubernetes clusters locally. Through a step-by-step guide, you learned how to deploy a basic Kubernetes cluster using minikube. Key Kubernetes components such as Pods, Deployments, Services, NodePort, and ConfigMaps were covered in detail, showcasing their essential roles in application deployment and management.

Hopefully you have now gained some useful insights into leveraging minikube to experiment with Kubernetes.

Wei-Meng Lee

Implementing the Outbox Pattern with Kafka and C#

The outbox pattern is a proven and scalable software design pattern often used in a distributed environment to publish events reliably and enforce data consistency. This article presents an overview of the outbox pattern and examines how it can be implemented using Kafka and C# in ASP.NET Core applications. If you're to work with the code examples discussed in, this

article you need the following installed in your system:

- Visual Studio 2022
- .NET 8.0
- ASP.NET Core 8.0 Runtime

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

In this article, I'll examine the following points:

- An overview of the outbox pattern and its benefits and downsides
- An introduction to Kafka as a message broker
- How the transaction outbox pattern works
- How to implement the outbox pattern in ASP.NET Core, Kafka, and C#
- How to implement the consumer application to consume messages stored in Kafka

A Real-World Use Case

Typically, microservices-based applications maintain their own data store to persist and retrieve data. This strategy helps improve the application's scalability and promotes agility in the deployment process. Additionally, these applications take advantage of messaging to facilitate asyn-

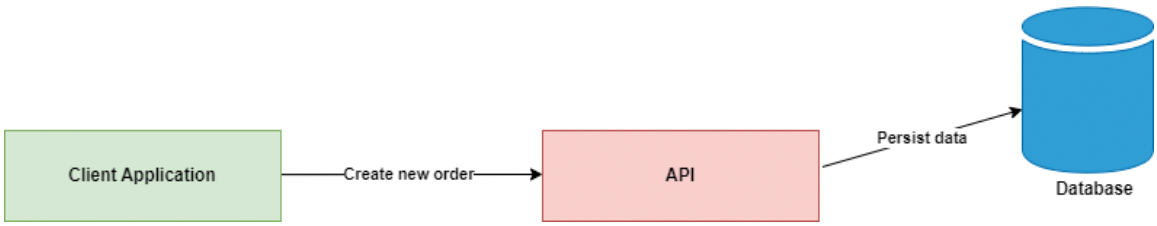

Figure 1: The flow of a typical Order Processing application

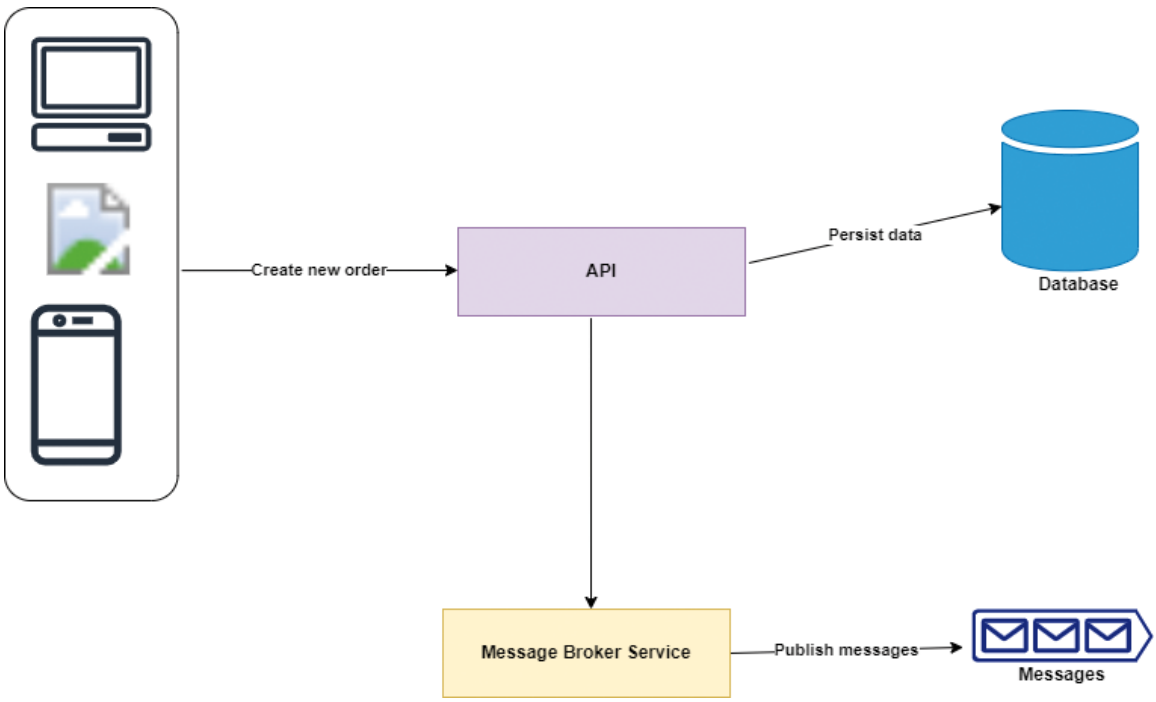

Figure 2: The API calls the message broker to publish the event.

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.

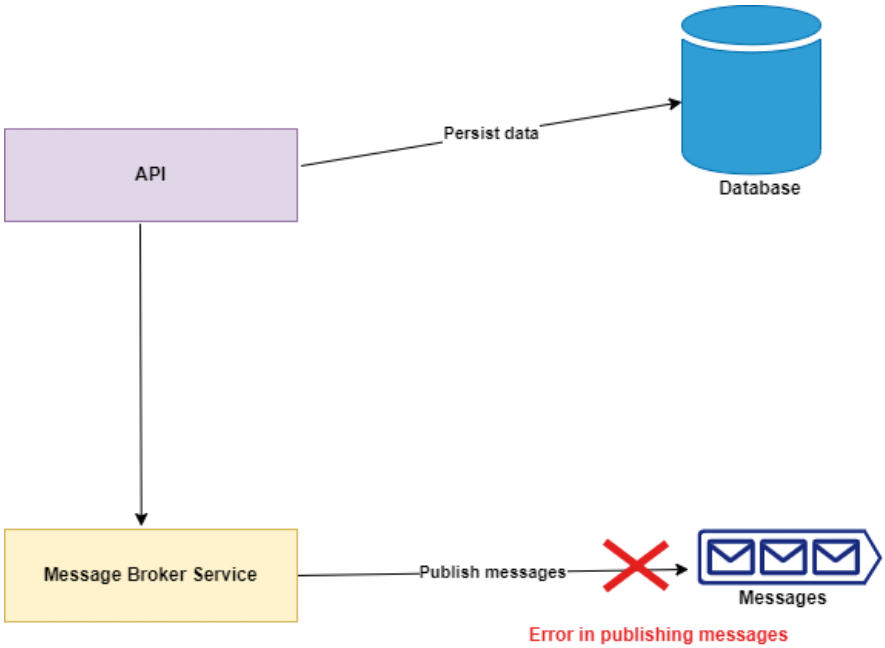

Figure 3: An error in publishing messages

chronous communication between the services. There are several messaging solutions for you to choose from such as Kafka, Azure Service Bus, RabbitMQ, etc.

Understanding the Problem

Consider the flow of a typical Order Processing application at a glance:

1. The client application sends a request to the API to persist the data to the underlying database.
2. The API validates the incoming request, executes the necessary business logic, and persists the data in the database, as shown in **Figure 1**.

3. Once the data is stored in the database, the API calls a message broker to publish the event so that other services can consume it if needed, as shown in **Figure 2.**

What's wrong with this? Although the flow may appear straightforward, complications can arise. For instance, if the message broker is unavailable, there's an error in publishing the messages or a runtime exception occurs in the application. As shown in **Figure 3**, the ordering service and the data can become inconsistent.

Addressing this issue can be challenging, as writing code to solve it may lead to a violation of the Single Responsibility Principle, potentially requiring additional responsibilities for the OrderService. Here's where the transactional outbox pattern comes to the rescue.

An Overview of the Outbox Pattern

The outbox pattern is a technique used in microservices architectures to guarantee reliable communication between services. For this purpose, a temporary storage area is created in the database of a microservice, often referred to as an **outbox table**, in which the outgoing data changes are recorded. The data is serialized before being sent from the outbox table to other external systems or services. In distributed systems, duplicate writes occur when an operation must update a local database and notify other services.

The outbox design solves this problem by combining the publication of messages/events with the same database transaction that updates the local state—making both actions atomic and, therefore, complete or incomplete. In the outbox pattern, you won't be publishing an event to the message broker directly. Instead, a record will be inserted into a special database named Outbox. Incidentally, the Outbox database table is stored in the same database where the data of the application is being stored. Then an asynchronous process executes in the background and periodically checks for incoming messages. When it finds one, it publishes the message to a message broker (i.e., Apache Kafka, RabbitMQ, etc.), as shown in **Figure 4**.

Components of the Outbox Pattern

The outbox pattern is comprised of the following key components:

4. **Events:** At runtime, an application generates events that are stored in the Outbox table.
5. **Outbox table:** This data store is used to store the messages until they are processed by the publisher.
6. **Publisher:** This component is used to process messages from the outbox table and then dispatch them to the appropriate destinations

Benefits and Downsides

The outbox pattern is a proven technique widely used for enhancing data consistency and reliability in microservices architectures. However, like any architectural pattern, it has its benefits and downsides.

Benefits

The following are the benefits of the outbox pattern:

- **Transactional consistency:** The outbox pattern ensures that any changes to the database residing locally and the publishing of events/messages are part of the same database transaction. This prevents data inconsistencies that might occur due to operation failures after the database changes are committed but before the events are published. By sending the messages as part of the same transaction, the outbox pattern maintains atomicity and consistency. Moreover, the outbox pattern enables events to be retried until they have been successfully processed.
- **Reliable message publishing:** The outbox pattern ensures reliable message sending by taking advantage of a persistent storage mechanism where the messages or events are stored in the database, thereby preventing message loss due to failures in the event publishing mechanism or network issues. This persistent storage of messages coupled with asynchronous message sending helps decouple the message dispatch process from the application's response, thereby minimizing the chances of message loss and improving resiliency.

- **Scalability:** The outbox pattern enhances scalability of an application by separating the concerns of event creation from event distribution. It enables an application to focus on processing incoming requests without interruption by offloading message transmission to a different process or service.
- **Performance:** The outbox pattern improves performance by sending messages asynchronously, thereby processing the messages without any delays to enhance responsiveness and throughput. Storing the events in the same database transaction as the business operations minimizes the number of total transactions executed, which can enhance performance.

Downsides

Despite the benefits of the outbox pattern stated earlier, there are certain downsides as well:

- **Increased complexity:** The outbox pattern can introduce additional complexity to your architecture, application design, and development. By implementing this pattern, you need to manage the outbox processor and verify whether the messages are recorded correctly, thereby making your application difficult to manage, maintain, and debug.
- **Latency:** When using the outbox pattern, there might be a delay from the time an event occurs and when it's published to other services. It usually publishes events in batches at a regular time interval set by the outbox processor. However, this delay isn't recommended in situations where an immediate or real-time communication is needed.
- **Database coupling:** Because the outbox pattern depends on the database to store events before they're published, there's an inherent coupling with the database, thereby making future changes more challenging.
- **Scalability:** Although the outbox pattern ensures that microservices are not tightly coupled and can be independently scaled, scaling your application may still be challenging. If an outbox processor isn't working as expected or there's too much traffic, you might have to consider scaling your infrastructure for processing outboxes.
- **Resiliency:** Although the outbox pattern ensures that events aren't lost, there's no built-in mechanism to handle failures while processing the events. You need to write your code to handle duplicate events, idempotency, retries, and failure recovery.
- **Operational overhead:** Monitoring the outbox repository regularly and adopting the strategy to transmit messages and potential retries can be an additional overhead. Keeping an eye on the outbox storage, ensuring that the messages are delivered, and dealing with errors or retries may be challenging.
- **Integration challenges:** You must take the necessary steps to ensure reliable message delivery. This may involve implementing additional configurations and dependencies when connecting to external systems like RabbitMQ and Apache Kafka.

How Does the Outbox Pattern Work?

Assume an order processing application in which an order service is used to handle orders placed by the customer. As soon as a new order is placed, the application must update the order database table and also notify the inventory service that the stock should be updated. The complete flow is illustrated in the sequence of steps of a typical order processing use case given below:

1. **Create Order:** A customer uses the application to place a new order. Next, the Order Service picks up this order and processes it.
2. **Update Order database table:** The Order Service stores the new order in the Order database table with information that includes the order details and the customer details together with the Order status.
3. **Create Stock Update message:** Besides updating the Order database table, the Order Service creates a message to indicate that the Stock database table needs to be updated. This message is persisted in the Outbox database table within the context of the same transaction.
4. **Commit transaction:** Next, the transaction that comprises updating the Order database table and insertion of the message into the Outbox database table is committed.
5. **Check Outbox database table periodically:** A process named Message Relay executes in the background to check the Outbox database table for new messages.
6. **Dispatch message to Stock service:** When the Message Relay detects a new message in the Outbox database table, it forwards the message to the Stock service for processing via the Message queue.
7. **Stock update:** Upon receipt of a new message, the Stock service updates the Stock database table accordingly.

In the outbox pattern, when an event occurs, instead of a message being sent to a message broker such as Kafka, a new record that contains the details of the event is stored in the Outbox database table. It should be noted that typically the Outbox table is part of the same data-

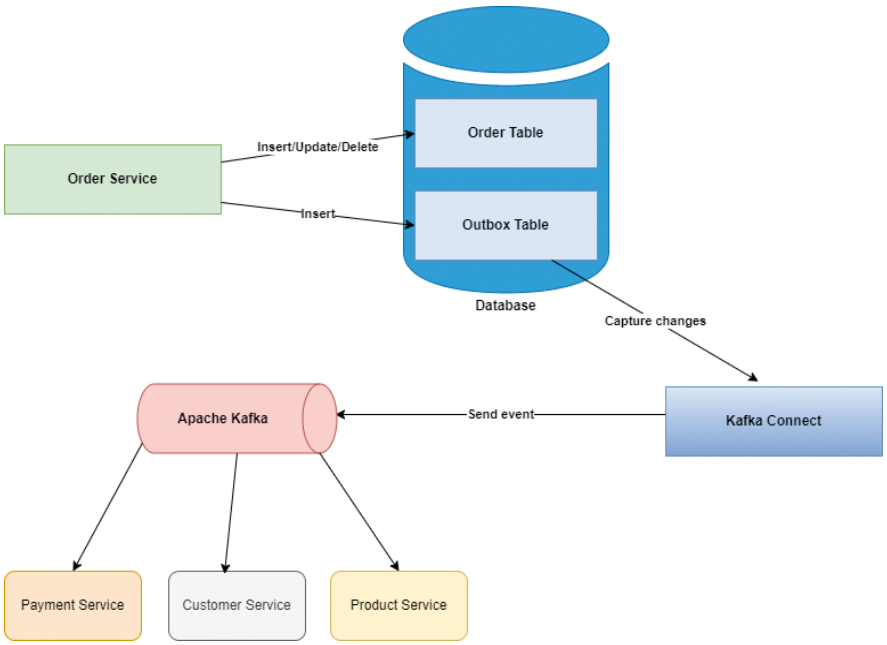

Figure 4: Demonstrating the Outbox Pattern

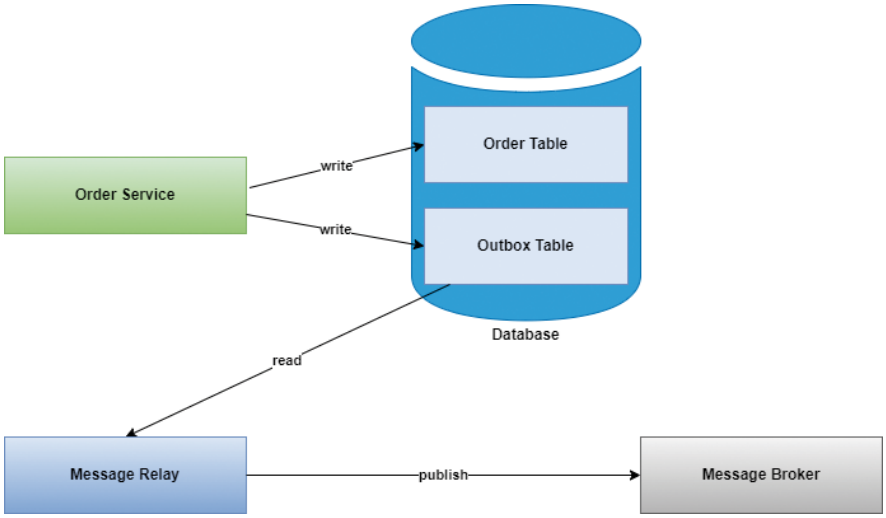

Figure 5: The Outbox Pattern in action

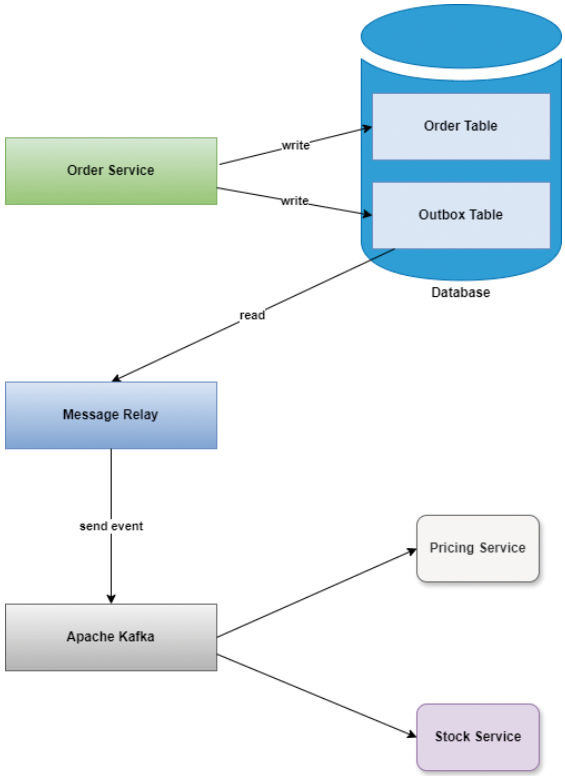

Figure 6: Using Kafka as the message broker in a typical Outbox pattern implementation

base and this all happens as part of the same transaction. **Figure 5** illustrates how this all works together.

Key Terminologies

Here are the key terminologies you often come across when exploring the Outbox pattern:

- **Event log or the Outbox:** Each microservice maintains an event log or an Outbox where the events that have to be published are first recorded to ensure that the events are never lost. Essentially, an Outbox is a database table to store events or messages before the events are published.
- **Transactional operation:** For the purpose of ensuring that only successful business operations are captured in the event log, events are added to the log as part of the same transaction as the business operations that generated them.
- **Outbox processor:** An outbox processor runs as a separate background task and scans the event log at regular intervals of time. It collects events without processing them and then publishes them to the event bus or message broker.
- **Message broker or event bus:** A message broker or an event bus receives those events that have been published. The message broker or event bus provides durability and reliability and distributes these events to the subscribers or the other microservices.

Introduction to Kafka as a Message Broker

Apache Kafka is a distributed, high-performance, open source, scalable, and versatile stream-processing software. It's written in Java and Scala, and has become popular in recent times for building systems that are adept at handling massive volumes of data. Kafka is a messaging platform based on the publish/subscribe model. It comes with built-in features for replication, partitioning, fault tolerance, and improved throughput, compared to applications without message brokers.

Kafka is a good option for applications that need vast data processing capabilities. The main use of Kafka is in creating real-time streaming data pipelines. For this reason, Kafka includes stream processing and fault-tolerant storage functionalities, which enable storing and analyzing historical and real-time data. **Figure 6** shows how Kafka fits in a typical implementation of the outbox pattern.

Getting Started with Apache Kafka

To get started with Apache Kafka, follow the steps outlined in the next few sections.

Download and Extract Kafka

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

Alternatively, you can run the following command at the terminal to download Kafka in your computer:

```
wget https://archive.apache.org/
dist/kafka/3.7.0/
kafka_2.13-3.7.0.tgz
```

Once Kafka has been downloaded, you should extract the zip archive by running the following command at the terminal window:

```
tar -xzf kafka_2.13-3.7.0.tgz
```

Start ZooKeeper

You need ZooKeeper to run Kafka on your computer. Once you've downloaded and extracted Apache Kafka to your computer, start ZooKeeper by running the following command at the terminal window:

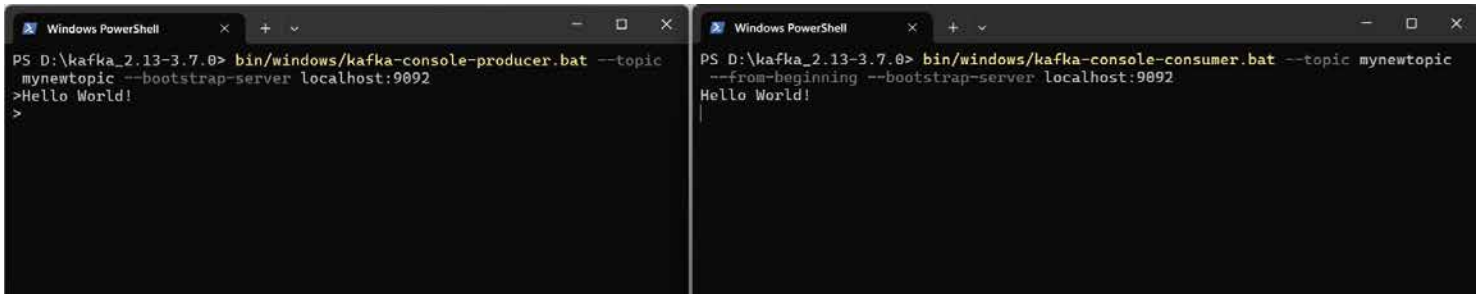

Figure 7: The Kafka consumer in action

```
cd kafka_2.13-3.7.0
bin/windows/
zookeeper-server-start.bat
config/zookeeper.properties
```

Create a Topic

Now that ZooKeeper and Kafka are up and running in your computer, you can start creating topics. You can create a new topic by running the following command at the terminal window.

```
.\bin\windows\
kafka-topics.bat --create
--topic mynewtopic
--bootstrap-server
localhost:9092
```

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
mynewtopic --from-beginning
```

Display all Topics

To list all topics, you can use the following command:

```
.\bin\windows\kafka-topics.bat
--list --bootstrap-server
localhost:9092
```

Start the Producer

You can run the following command at the terminal window to start a producer.

```
bin/windows/
kafka-console-producer.bat
--topic mynewtopic
--bootstrap-server
localhost:9092
```

Start the Consumer

In Kafka, you need a consumer to read the messages produced by a producer. You can run the following command at the terminal window to read all messages sent by the producer as shown in **Figure 7**.

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

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 use the kafka-server-stop.bat script as shown below:

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

Implementing the Outbox Pattern in ASP.NET Core, Kafka, and C#

In this section, I'll examine how to implement the outbox pattern in an ASP.NET Core application using Kafka. In this example, you'll use the following interfaces and classes.

- **OutboxMessage:** This is the model class
- **IKafkaProducer:** This is the interface for the KafkaProducer class.
- **IKafkaConsumer:** This is the interface for the KafkaConsumer class.
- **IOrderService:** This interface contains the declaration of methods supported by the OrderService.
- **IOutboxMessageRepository:** This represents the interface for the OutboxMessageRepository class.
- **KafkaProducer:** This class is used to send messages to Kafka.
- **KafkaConsumer:** This class is used to consume messages in Kafka.
- **OrderService:** The OrderService class implements the IOrderService interface and encapsulates the logic for invoking the appropriate methods of the OrderMessageRepository.
- **KafkaMessageProcessor:** This is a background service that calls KafkaConsumer at regular intervals of time to consume messages residing in Kafka.
- **OutboxMessageProcessor:** This is yet another background service used to invoke KafkaProducer to transmit messages to Kafka.
- **OutboxMessageRepository**: This class contains methods to retrieve messages from the Outbox database table and also update a message as needed.

- **ApplicationDbContext:** This class represents the data context that acts as a gateway to connect to the underlying database being used by the application.
- **OrderController:** This represents the Order API that can be consumed by clients to retrieve existing orders or create new orders.

Create a New ASP.NET Core 8 Project in Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways such as, from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. 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.

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 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 Outbox_Pattern_Demo 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," "Do not use top-level statements", and the "Enable OpenAPI support" checkboxes are unchecked because you won't use any of these in this example.
6. Remember to leave the **Use controllers** checkbox checked because you won't use minimal API in this example.
7. Click Create to complete the process.

A new ASP.NET Core Web API project is created. You'll use this project to implement the outbox pattern using Kafka in ASP.NET Core and C#.

Create the Outbox Database Table

First off, create the Outbox database table to store messages. To do this, create a new database called Outbox_Pattern_Demo using the database script given below:

```
CREATE DATABASE Outbox_Pattern_Demo;
```

Next, create a new database table called OutboxMessage using the following database script.

```
CREATE TABLE Outbox_Message
(
   Event_Id BIGINT IDENTITY PRIMARY KEY,
   Event_Payload NVARCHAR(MAX) NOT NULL,
   Event_Date DATETIME NOT NULL
   DEFAULT CURRENT_TIMESTAMP,
   IsMessageDispatched BIT NOT NULL
);
```

Create another database table in the same database named Order using the following script.

```
CREATE TABLE [Order]
(
    Order_Id BIGINT IDENTITY PRIMARY KEY,
    Customer_Id INT NOT NULL,
    Order_Date DATETIME,
    Order_Amount MONEY NOT NULL
);
```

Install NuGet Package(s)

In this example, you'll take advantage of Entity Framework to interact with the database and perform CRUD (an acronym for Create Read Update Delete) operations. You'll also need a Kafka client package to produce and consume messages. To work with Apache Kafka, you'll use the Confluent.Kafka NuGet package. To install the required package into your project, right-click on the solution and then select Manage NuGet Packages for Solution…. Now search for the Microsoft.EntityFramework.SqlServer and Confluent.Kafka packages in the search box and install it. Alternatively, you can type the commands shown below at the NuGet Package Manager Command Prompt:

```
PM> Install-Package
Microsoft.EntityFrameworkCore.SqlServer
Install-Package Confluent.Kafka
```

You can also install this package by executing the following commands at the Windows Shell:

```
dotnet add package
Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Confluent.Kafka
```

Create the OutboxMessage Class

Create a new C# class named OutboxMessage in a file named OutboxMessage.cs and write the following code in there.

```
public class OutboxMessage
{
  public int Event_Id { get; set; }
  public string Event_Payload { get; set; }
  public DateTime Event_Date { get; set; }
  public bool IsMessageDispatched { get; set; }
}
```

Create the Order Model Class

Create a new class named Order in a file having the same name with a .cs extension and write the following code in there:

```
public class Order
{
   public long Order_Id { get; set; }
   public int Customer_Id { get; set; }
   public DateTime Order_Date { get; set; }
   public decimal Amount { get; set; }
}
```

Create the Data Context

In Entity Framework Core (EF Core), a data context is a component used by an application to interact with the database and manage database connections, and to query and persist data in the database. You'll now create a data context class named CustomerDbContext. To create a data context, create a class that extends the DbContext class of EF Core, as shown below:

```
public class ApplicationDbContext :
             DbContext
{
    public ApplicationDbContext
   (DbContextOptions
   <ApplicationDbContext>
    options) : base(options)
    {
    }

    protected override void
    OnModelCreating
    (ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
    }

    public DbSet<OutboxMessage>
    OutboxEvents { get; set; }
    public DbSet<Order>
    Orders { get; set; }
}
```

You should specify the primary keys and the table names for your entities in the OnModelCreating method, as shown in the code snippet given below:

```
protected override void
OnModelCreating
(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Order>(entity =>
    {
        entity.ToTable("Order");
        entity.HasKey(e => e.Order_Id);
    });

    modelBuilder.Entity<OutboxMessage>
    (entity =>
    {
        entity.ToTable("OutboxMessage");
        entity.HasKey(e => e.Event_Id);
    });

    base.OnModelCreating(modelBuilder);
}
```

The complete source of the data context class is given in **Listing 1**.

Create the OrderService Class

Now, create a new class named OrderService in a file having the same name with a .cs extension. Write the following code in there:

```
public class OrderService :
IOrderService
{

}
```

The OrderService class illustrated in the code snippet below implements the methods of the IOrderService interface. Here's how the IOrderService interface should look:

```
public interface IOrderService
{
   public Task<List<Order>>
   GetAllOrdersAsync();
   public Task<Order>
   GetOrderAsync(int Id);
   public Task
   CreateOrderAsync(Order order);
}
```

The OrderService class implements the methods of the IOrderService interface.

```
public async Task
CreateOrderAsync(Order order)
{
using var transaction =
 _context.Database.BeginTransaction();

    try
    {
        _context.Orders.Add(order);
        await _context.SaveChangesAsync();

        var outboxMessage = new OutboxMessage
        {
            Event_Payload =
            JsonSerializer.Serialize(order),
            Event_Date = DateTime.Now,
            IsMessageDispatched = false
        };

        _context.OutboxMessages.
        Add(outboxMessage);

        await _context.SaveChangesAsync();
        await transaction.CommitAsync();
```

Listing 1: The ApplicationDbContext class

```
using Microsoft.EntityFrameworkCore;

namespace Outbox_Pattern_Demo
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions
        <ApplicationDbContext> options) : base(options)
        {
        }

        protected override void OnModelCreating
        (ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Order>(entity =>
            {
                entity.ToTable("Order");
                entity.HasKey(e => e.Order_Id);
            });

            modelBuilder.Entity
            <OutboxMessage>(entity =>
            {
                entity.ToTable("OutboxMessage");
                entity.HasKey(e => e.Event_Id);
            });

            base.OnModelCreating(modelBuilder);
        }

        public DbSet<OutboxMessage>
        OutboxMessages { get; set; }
        public DbSet<Order>
        Orders { get; set; }
    }
}
```

Listing 2: The OrderService Class

```
namespace Outbox_Pattern_Demo
{
   using System.Text.Json;

    public class OrderService : IOrderService
    {
        private readonly ApplicationDbContext _context;

        public OrderService(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<List<Order>> GetAllOrdersAsync()
        {
            return await Task.FromResult
            (_context.Orders.
            ToList<Order>());
        }

        public async Task<Order>
        GetOrderAsync(int Id)
        {
            return await Task.FromResult
            (_context.Orders.FirstOrDefault
            (x => x.Order_Id == Id));
        }

        public async Task CreateOrderAsync(Order order)
        {
            using var transaction =
            _context.Database.BeginTransaction();

            try
            {
                _context.Orders.Add(order);
                await _context.SaveChangesAsync();

                var outboxMessage = new OutboxMessage
                {
                    Event_Payload =
                    JsonSerializer.Serialize(order),
                    Event_Date = DateTime.Now,
                    IsMessageDispatched = false
                };

                _context.OutboxMessages.
                Add(outboxMessage);

                await _context.SaveChangesAsync();
                await transaction.CommitAsync();
            }
            catch
            {
                await transaction.RollbackAsync();
                throw;
            }
        }
    }
}
```

```
    }
    catch
    {
        await transaction.RollbackAsync();
        throw;
    }
}
```

The complete source code of the OrderService class is given in **Listing 2.**

Create the OutboxMessageProcessor Background Service

You'll now create a service that runs in the background and checks for messages in the OutboxMessage database table. If one or more messages are found, it sends them to Kafka for further processing. To create a background service in .NET Core, you can take advantage of the IHostedService interface. You can create your background service class by implementing the IHostedService interface.

This interface contains two methods, StartAsync and StopAsync as shown below:

```
public interface IHostedService
{
    Task StartAsync
    (CancellationToken cancellationToken);

    Task StopAsync
    (CancellationToken cancellationToken);
}
```

You can also use the abstract helper class called BackgroundService that's available in .NET Core. Because this class already implements the IHostedService interface, you can create your background service class by extending this class in lieu of implementing the IWorkerService interface. The BackgroundService class is defined in the Microsoft.Extensions.Hosting namespace as shown below:

```
public abstract class
BackgroundService :
IHostedService,
IDisposable
{
    public virtual void Dispose();
    public virtual Task
    StartAsync(CancellationToken
    cancellationToken);
    public virtual Task
    StopAsync(CancellationToken
    cancellationToken);
    protected abstract Task
    ExecuteAsync(CancellationToken
    stoppingToken);
}
```

The following code snippet shows how you can create your background service class in this way.

```
public class OutboxMessageProcessor :
BackgroundService
{
   protected override async Task
   ExecuteAsync(CancellationToken
   cancellationToken)
   {

   }
}
```

The PublishOutboxMessagesAsync method of the OutboxMessageProcessor class is responsible for sending the outbox messages to Kafka. The foreach loop iterates through all messages residing in the Outbox database table that are yet to be dispatched. Such messages are sent to Kafka

Listing 3: The OutboxMessageProcessor class

```
namespace Outbox_Pattern_Demo
{
    public class OutboxMessageProcessor : BackgroundService
    {
        private readonly IServiceScopeFactory _scopeFactory;
        private readonly IKafkaProducer _producer;

        public OutboxMessageProcessor
        (IServiceScopeFactory scopeFactory,
        IKafkaProducer producer)
        {
            _scopeFactory = scopeFactory;
            _producer = producer;
        }

        protected override async Task ExecuteAsync
        (CancellationToken cancellationToken)
        {
            while (!cancellationToken.IsCancellationRequested)
            {
                await PublishOutboxMessagesAsync
                (cancellationToken);
            }
        }

        private async Task PublishOutboxMessagesAsync
        (CancellationToken cancellationToken)
        {
            try
            {
                using var scope = _scopeFactory.CreateScope();
                await using var _dbContext =
                scope.ServiceProvider.GetRequiredService
                <ApplicationDbContext>();

                List<OutboxMessage> messages =
                 _dbContext.OutboxMessages.Where
                (om => om.IsMessageDispatched != false).ToList();

                foreach (OutboxMessage outboxMessage in messages)
                {
                    try
                    {
                        await _producer.
                        SendMessageToKafkaAsync(outboxMessage);

                        outboxMessage.IsMessageDispatched = true;
                        outboxMessage.Event_Date = DateTime.UtcNow;

                        _dbContext.OutboxMessages.
                        Update(outboxMessage);
                        await _dbContext.SaveChangesAsync();
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine(e);
                    }
                }
            }
            catch
            {
                throw;
            }

            await Task.Delay
            (TimeSpan.FromSeconds(5),
            cancellationToken);
        }
    }
}
```

by making a call to the SendMessageToKafkaAsync method of the KafkaProducer class.

```
private async Task
PublishOutboxMessagesAsync
(CancellationToken cancellationToken)
{
    try
    {
        using var scope =
       _scopeFactory.CreateScope();
        await using var _dbContext =
        scope.ServiceProvider.
        GetRequiredService
        <ApplicationDbContext>();

        List<OutboxMessage> messages =
        _dbContext.OutboxMessages.Where(om =>
        om.IsMessageDispatched
        != false).ToList();

        foreach (OutboxMessage outboxMessage
        in messages)
        {
            try
            {
                await _producer.
                SendMessageToKafkaAsync
                (outboxMessage);

                outboxMessage.
                IsMessageDispatched = true;
                outboxMessage.Event_Date =
                DateTime.UtcNow;

                _dbContext.OutboxMessages.
                Update(outboxMessage);
                await _dbContext.
                SaveChangesAsync();
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
    catch
    {
        throw;
    }

    await Task.Delay
    (TimeSpan.FromSeconds(5),
    cancellationToken);
}
```

The complete source code of the OutboxMessageProcessor class is given in **Listing 3**.

Create the OutboxMessageRepository

Let's now create the OutboxMessageRepository that will be used to get the messages from the Outbox database table and also update a message in the same table. The OutboxMessageRepository implements the IOutboxMessageRepository interface. The following code snippet shows the IOutboxMessageRepository interface:

```
namespace Outbox_Pattern_Demo
{
    public interface
```

Listing 4: The OutboxMessageRepository class

```
using System.Collections.ObjectModel;
using System.Data;

namespace Outbox_Pattern_Demo
{
    public class OutboxMessageRepository :
    IOutboxMessageRepository
    {
        private readonly ApplicationDbContext _context;
        private IreadOnlyCollection
        <OutboxMessage> _outboxMessages;

        public OutboxMessageRepository
        (ApplicationDbContext context)
        {
            _context = context;
        }

        public IreadOnlyCollection
        <OutboxMessage> OutboxMessages
        {
            get
            {
                return _outboxMessages ??
                (_outboxMessages = new
                ReadOnlyCollection<OutboxMessage>
                (_context.OutboxMessages.ToList()));
            }
        }

        public async Task<IreadOnlyCollection
        <OutboxMessage>> GetUnsentMessagesAsync()
        {
            List<OutboxMessage>? unsentMessages =
            _context.OutboxMessages.Where
            (e => e.IsMessageDispatched != true).ToList();
            ReadOnlyCollection<OutboxMessage>?
                result = new ReadOnlyCollection
                <OutboxMessage>(unsentMessages);
                return result;
        }

        public async Task<IreadOnlyCollection
        <OutboxMessage>> GetMessagesByIdsAsync
        (IEnumerable<int> ids)
        {
            List<OutboxMessage>?
            orders = _context.
            OutboxMessages.ToList();
            var readOnlyOrders =
            new ReadOnlyCollection
            <OutboxMessage>(orders);
            return readOnlyOrders;
        }

        public async Task UpdateAsync
        (OutboxMessage message, bool status)
        {
            var entity = _context.OutboxMessages.
            FirstOrDefault(o => o.Event_Id == message.Event_Id);

            if (entity != null)
            {
                entity.Event_Id = message.Event_Id;
                entity.Event_Date = message.Event_Date;
                entity.Event_Payload = message.Event_Payload;
                entity.IsMessageDispatched =
                message.IsMessageDispatched;
                await _context.SaveChangesAsync();
            }
        }
    };
}
```

Listing 5: The OrderController Class

```
using Microsoft.AspNetCore.Mvc;

namespace Outbox_Pattern_Demo.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class OrderController : ControllerBase
    {
        private IOrderService _orderService;
        public OrderController
        (IOrderService orderService)
        {
            _orderService = orderService;
        }

        [HttpGet("GetOrders")]
        public async Task<List<Order>> GetOrders()
        {
            return await _orderService.
            GetAllOrdersAsync();
        }

        [HttpGet("{id}")]
        public async Task<Order> GetOrder(int id)
        {
            return await _orderService.
            GetOrderAsync(id);
        }

        [HttpPost("CreateOrder")]
        public async Task<IActionResult>
        CreateOrder([FromBody] Order order)
        {
            await _orderService.
            CreateOrderAsync(order);
            return Ok();
        }
    }
}
```

```
    IOutboxMessageRepository
    {
        Task<IreadOnlyCollection
        <OutboxMessage>>
        GetUnsentMessagesAsync();
        Task<IreadOnlyCollection
        <OutboxMessage>>
        GetMessagesByIdsAsync
        (IEnumerable<int> ids);
        Task UpdateAsync
        (OutboxMessage message,
        bool status);
    };
}
```

The complete source code of the OutboxMessageRepository class is given in **Listing 4**.

The OrderController Class

Next, create a new API controller class named OrderController and replace the generated code with the code given in **Listing 5**. The OrdersController class contains three action methods: GetOrder, GetOrders, and CreateOrder. The GetOrders action method returns a list of Order instances, and the GetOrder action method returns one Order based on the Order ID passed to the action method as a parameter.

The CreateOrder action method given below creates a new order record in the order database table.

```
[HttpPost("CreateOrder")]
public async Task<IActionResult>
```

Listing 6: The KafkaProducer class

```
using Confluent.Kafka;
using System.Net;

namespace Outbox_Pattern_Demo
{
    public class KafkaProducer : IKafkaProducer
    {
        private readonly ProducerConfig _producerConfig;
        private readonly IOutboxRepository _outboxRepository;
        private readonly string topic = "test";

        public KafkaProducer(IOutboxRepository outboxRepository)
        {
            _outboxRepository = outboxRepository;

            _producerConfig = new ProducerConfig
            {
                BootstrapServers = "localhost:9092",
                ClientId = Dns.GetHostName()
            };

            _outboxRepository = outboxRepository;
        }

        public async Task SendMessageToKafkaAsync
        (OutboxMessage message)
        {
            if (message == null)
            {
                throw new ArgumentNullException
                    (nameof(message));
            }

            using var producer =
            new ProducerBuilder<Null,
            string>(_producerConfig).Build();

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

                if (result.Status == PersistenceStatus.Persisted)
                {
                    await _outboxRepository.UpdateAsync
                    (message, OutboxMessageStatus.Sent);
                }
            }
            catch (Exception)
            {
                await _outboxRepository.UpdateAsync
                (message, OutboxMessageStatus.New);
            }
        }
    };
}
```

```
CreateOrder([FromBody] Order order)
{
    await _orderService.
    CreateOrderAsync(order);
    return Ok();
}
```

The GetOrders and GetOrder action methods call the GetOrdersAysnc and GetOrderAsync methods of the OrderService class respectively. The CreateOrder method calls the CreateOrderAsync method of the OrderService class and passes an instance of the Order class as the only parameter. Note how an instance of type IOrderService is injected into the OrdersController using constructor injection.

Send Messages to Apache Kafka

Create an interface named IKafkaProducer in a file named IKafkaProducerr.cs and write the following code in there:

```
public interface IKafkaProducer
{
   Task SendMessageToKafkaAsync
   (OutboxMessage message);
};
```

Next, create a new class named KafkaProducer in a file having the same name with a .cs extension and write the code given in **Listing 6** in there.

Register the Service Instances with IServiceCollection

The following code snippet illustrates how an instance of type IOrderService is added as a scoped service to the IServiceCollection.

```
builder.Services.AddScoped
<IOrderService, OrderService>();
```

Listing 7: The Program.cs file (Producer Application)

```
using Microsoft.EntityFrameworkCore;
using Outbox_Pattern_Demo;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddScoped
<IOrderService, OrderService>();
builder.Services.AddScoped
<IOutboxMessageRepository,
OutboxMessageRepository>();
builder.Services.AddScoped
<IKafkaProducer, KafkaProducer>();

builder.Services.AddSingleton
<IHostedService, OutboxMessageProcessor>();

builder.Services.AddDbContext
<ApplicationDbContext>(options =>
options.UseSqlServer
(@"Data Source=JOYDIP;
Initial Catalog=Outbox_Pattern_Demo;
Trusted_Connection=True;
TrustServerCertificate=True;
Integrated Security=True;"));

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseAuthorization();
app.MapControllers();
app.Run();
```

In addition, you should register the instances of type ImessageRepository and IKafkaProducer with the service collection, as shown below:

```
builder.Services.AddScoped
<IOutboxMessageRepository,
OutboxMessageRepository>();
```

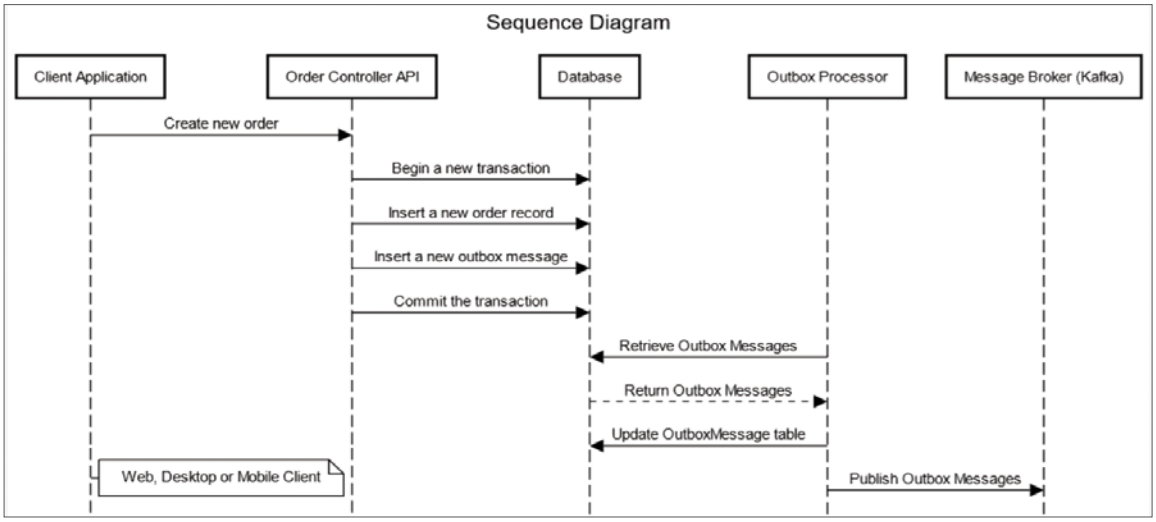

Figure 8: A sequence diagram of the complete flow

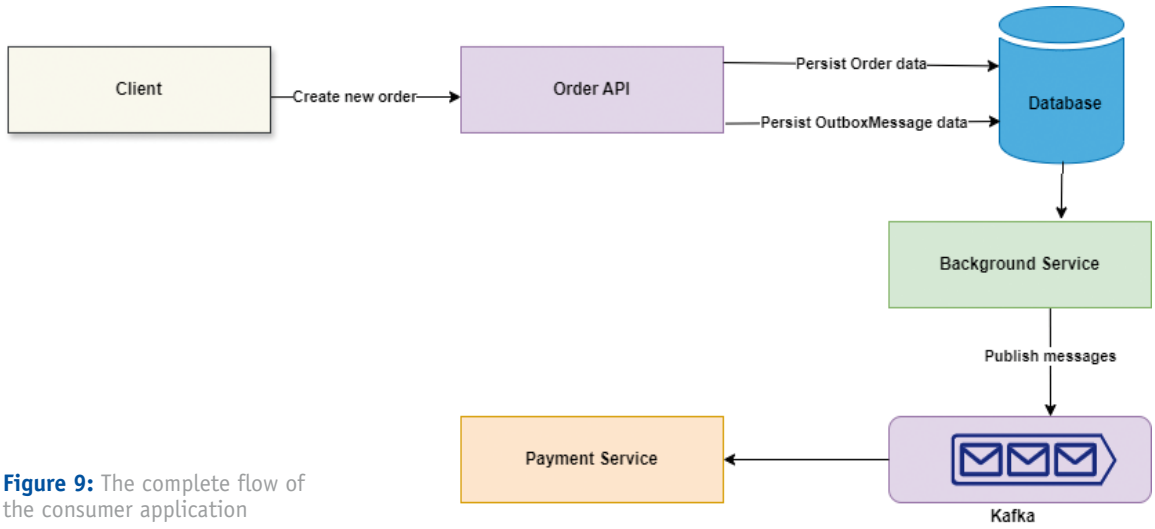

Figure 9: The complete flow of the consumer application

```
builder.Services.AddScoped
<IKafkaProducer, KafkaProducer>();
```

You should also register the hosted service in the Program.cs file, as shown in the code snippet given below:

```
services.AddSingleton
<IHostedService,
OutboxMessageProcessor>();
```

The complete source code of the Program.cs file is given in **Listing 7**.

Sequence Diagram of the Complete Flow

The sequence diagram of the complete flow is shown in **Figure 8**.

Create the Message Consumer Application

Let's create a message consumer to consume the messages stored in Kafka. To do this, create another ASP.NET Core application by following the steps mentioned earlier in this article. Create a new interface named IKafkaConsumer and write the following code in there:

```
namespace Outbox_Pattern_Demo
{
    public interface IKafkaConsumer
    {
        public Task ConsumeMessagesAsync
        (CancellationToken cancellationToken);
    }
}
```

Next, create a new class named KafkaConsumer that implements the IKafkaConsumer interface, as shown in **Listing 8**. You now need a service that runs in the background and consumes the messages residing in Kafka. To do this, create a class named KafkaMessageProcessor that extends the BackgroundService class, as shown in **Listing 9**.

Register the instance of type IKafkaConsumer as a scoped service and the instance of KafkaMessageProcessor as a singleton hosted service.

```
builder.Services.AddScoped
<IKafkaConsumer, KafkaConsumer>();
builder.Services.AddSingleton
<IHostedService, KafkaMessageProcessor>();
```

Figure 9 illustrates the complete flow visually.

Listing 10 shows the Program.cs file of the consumer application.

Execute the Application

Run the producer and the consumer applications separately because they're part of different solutions. You should also set appropriate breakpoints in the source code of both applications so that you can debug them. To run the application, follow the steps outlined below:

1. Start ZooKeeper in a terminal window.
2. Start Kafka in another terminal window.
3. Execute the producer application.
4. Execute the consumer application.
5. Launch the Postman Http Debugger tool.
6. Send an HTTP POST request to the producer API using Postman.

Figure 10 shows how you can invoke the API endpoint in Postman.

Real-World Use Cases for the Outbox Pattern

The outbox pattern has several use cases in real-life: real-time notifications and data synchronization. The next two sections discuss these.

Real-Time Notifications

You might often want real-time notifications in your application to send instant alerts, messages, etc. You can use the outbox pattern to implement notifications in your application in real-time. These events can then be processed asynchronously, thereby triggering real-time notifications to the logged-in users of the application.

Data Synchronization

Real-time data synchronization is yet another use case where the outbox pattern fits in. For example, in an e-commerce application when a customer places an order, both the order and the stock database tables are updated simultaneously in the same transaction. Additionally, the

Listing 8: The KafkaConsumer class

```
using Confluent.Kafka;
using System.Text.Json;

namespace Outbox_Pattern_Demo
{
    public class KafkaConsumer : IKafkaConsumer
    {
        private readonly string topic = "test";
        private readonly string groupId = "test_group";
        private readonly string bootstrapServers =
        "localhost:9092";

        public async Task ConsumeMessagesAsync
       (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 order = JsonSerializer.Deserialize
                                <Order>(consumer.Message.Value);
                        }
                    }
                    catch (OperationCanceledException)
                    {
                        consumerBuilder.Close();
                    }
                }
            }
            catch
            {
                throw;
            }
        }
    }
}
```

Listing 9: The KafkaMessageProducer class

```
namespace Outbox_Pattern_Demo
{
    public class KafkaMessageProcessor : BackgroundService
    {
        private readonly IServiceScopeFactory _scopeFactory;
        private readonly IKafkaConsumer _consumer;

        public KafkaMessageProcessor
        (IServiceScopeFactory scopeFactory,
        IKafkaConsumer consumer)
        {
            _scopeFactory = scopeFactory;
            _consumer = consumer;
        }

        protected override async Task ExecuteAsync
        (CancellationToken cancellationToken)
        {
            while (!cancellationToken.IsCancellationRequested)
            {
                await _consumer.ConsumeMessagesAsync
                (cancellationToken);
            }
        }
    }
}
```

Listing 10: The Program.cs file (Consumer Application)

```
using Microsoft.EntityFrameworkCore;
using Outbox_Pattern_Client_Demo;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddScoped
<IKafkaConsumer, KafkaConsumer>();
builder.Services.AddSingleton
<IHostedService,
KafkaMessageProcessor>();

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseAuthorization();
app.MapControllers();
app.Run();
```

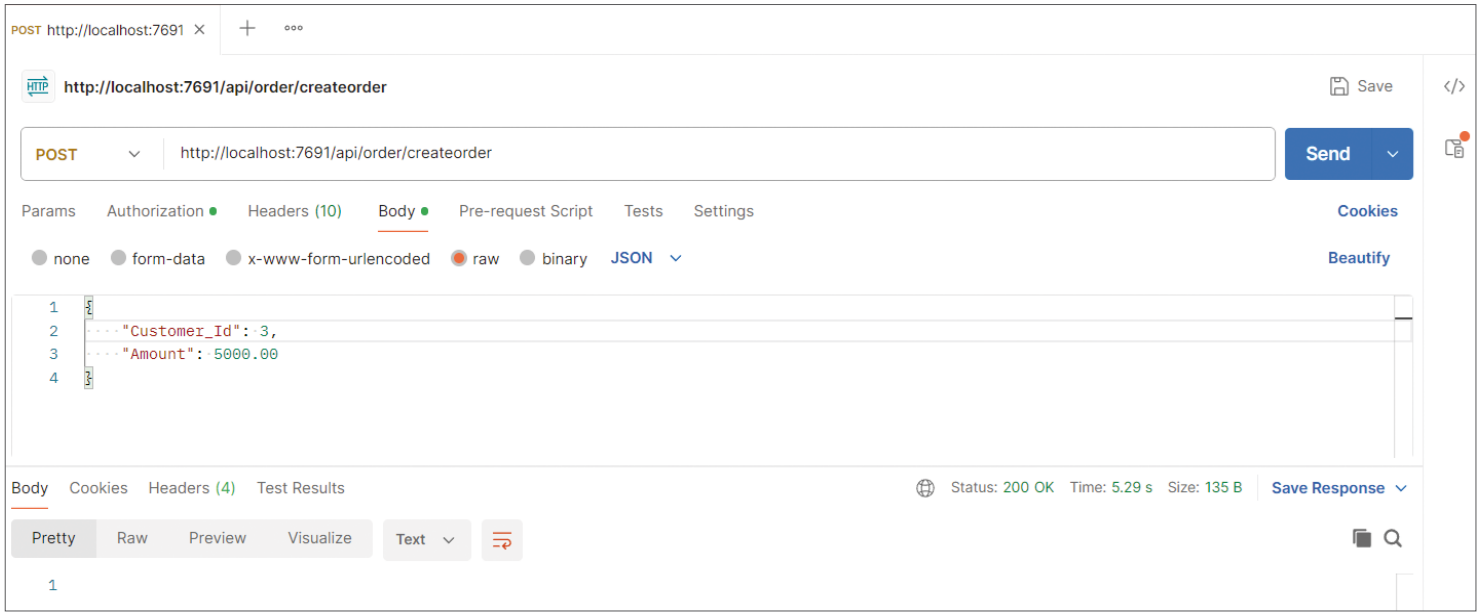

Figure 10: Invoking the API endpoint in Postman

order, product, and cart data must be synchronized across all devices in use using asynchronous processing.

Alternatives to the Transactional Outbox Pattern

Although the outbox pattern is a popular strategy for handling distributed transactions efficiently and ensuring reliable and consistent communication between microservices, there are certain downsides to using it as well. I've discussed them earlier in this article. The Two-Phase Commit and the Saga Pattern are two popular alternatives to this strategy.

Two-Phase Commit

This approach is used for performing an atomic transaction across multiple resources. Keep in mind that there are two phases in this strategy. While the transaction coordinator notifies all other resources about its desire to execute a transaction in the first phase, it instructs all resources to execute this operation in the next phase. The major drawback of such an approach is that if any resource fails or there are network issues, the whole transaction stops and locks are applied on resources.

Saga Pattern

This approach splits the transactions into multiple steps and completes each step as a separate transaction. This particular design is best suited for long-running transactions or those that take place between microservices. However, when you adopt this approach, it can be difficult to manage which transactions must be rolled back in the event of an error in your application. This is because the transaction timing and sequence for such transactions can be difficult to comprehend.

Where Do We Go from Here?

The outbox pattern, a reliable pattern for consistent event publishing in microservices-based applications, is a great way to make your microservices apps more stable and scalable. It helps separate services through which the events are successfully delivered and isolates business operations from publishing events. However, it also has certain downsides, which explains why you should understand these constraints before deciding on your application's design. I'll discuss more on the Outbox Pattern in a future article.

Joydip Kanjilal

Semantic Kernel Part 3: Advanced Topics

In the first article of this series, Semantic Kernel 101 in the January/February 2024 issue of CODE Magazine (https://codemag.com/Article/2401091/Semantic-Kernel-101), I gave an overview of the concepts of Semantic Kernel (SK), Microsoft's Framework for working with Large Language Models (LLMs). In the second article, Semantic Kernel 101: Part 2 in the March/April 2024 issue (https://codemag.com/Article/2403061/Semantic-Kernel-101-Part-2), I walked through hands-on examples of coding the basics with Semantic Kernal (SK). In this article, I'll create a truly useful real-world Copilot, walking through the evolution of my first professional AI application. I'll start at the beginning with the first version, showing every step our team took to get to a good result. Then I'll show you how, only a few months later, it's evolved into a much simpler and more effective app.

In the latest version, we allow the LLM to identify and call functions on its own without any of the step-by-step procedural code we used in the original version. I'll show you how we incorporated a very basic Retrieval Augmented Generation (RAG) pattern in the original version to ground the LLM's responses with customer data, and how the new version delegates even that work to the LLM.

The Original Implementation

The first time I was tasked with creating a "Copilot" system, I thought I had a pretty good understanding of LLMs and all the related technologies I'd need. In fact, I did have a pretty good conceptual understanding, but, like everyone else, neither I nor my team had much practical experience because LLMs were brand new. The goal of our new Copilot was to answer questions about a specified contract without the user being a legal expert or having to comb through the contract to find the answer. The user could just point to a contract and start asking questions in natural language. The first step would be to inject the entire text of a contract into the prompt, to give the LLM context.

> We created one simple prompt, and we were well on our way. Victory!

This worked surprisingly well and with little effort, our Copilot could answer a lot of questions. We created one simple prompt, and we were well on our way. Victory!

```
Based on the following:
#############
{{$contract}}
#############
Answer the question:
{{$input}}
```

LLMs were shiny and new and seemed nothing short of magic, but they're incapable of doing anything except generating text based on public data, mainly obtained from the internet, up to a certain date. By inserting the text of the contract into our prompt, we could extend the magic of the LLM to include the text of our contract, but only to a point. Our first real challenge came when we realized that there were still a lot of questions our Copilot couldn't answer. They seemed like simple questions, but they caused the Copilot to stumble. For instance, if a user asked whether a contract had expired, the LLM couldn't answer because the LLM wasn't trained on what today's date is. To solve this, we retrieved this additional data in code and injected it into the prompt:

```
{{$dateandtime}}
Based on the following:
#############
{{$contract}}
#############
Answer the question:
{{$input}}
```

We developed intricate ways of determining if the current date and time might be required to answer a question, so we could retrieve it and add it to the prompt, but we found it difficult to get right. We looked for keywords and phrases in the question to determine if the date and time was required to answer the question, but the approach was error prone. It turned out that at that time, the most successful way to determine if a piece of information was needed was to ask the LLM with a prompt like this one:

```
Based on the following:
#############
{{$input}}
#############
Would it help to know the current date
or time to answer the question?
Answer only with a single word YES or NO.
Do not expand on your answer.
```

This approach worked pretty well, because the LLM was now provided with the current date and time when it was relevant to the user's question. If it wasn't relevant, the {{$dateandtime}} placeholder was set to an empty string and didn't affect the prompt. Of course, we hand-coded all of this and we now had to make two calls to the LLM for every question. In fact, we'd have to hand-code something similar and make an additional call for every piece of information that might be useful in answering a user's question. For instance, what's the user's current location? We could use Location Services in Windows to retrieve the user's location, but we had to ask the LLM if the location was helpful for each question.

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.

For every bit of information, we created a prompt to ask the LLM if it was relevant to the current question and send it. If the info was useful, we'd retrieve the data and merge it into the prompt. At one point, we thought it couldn't hurt to just always inject all this information into every prompt so we could skip all those preliminary calls, but it turned out that, aside from being slow including a lot of extraneous information, doing this often degraded the quality of the response. The code quickly became quite ugly, but it worked! After the text of the contract and all the potentially useful information was added to the prompt, the prompt became very large, which made processing the prompt slow. In some cases, the prompt became too large to run at all, and sometimes it became quite convoluted, which made it more difficult to get good answers from the LLM. But it mostly worked!

The size of the larger contracts was a problem we had to solve. Even though it was built to handle only one contract at a time, the system could only handle contracts stored in plain text only up to a certain size. One of the sample contracts our testers provided to us after our early successes was a 630-page PDF, and we realized that we had two problems. First, we had no way of inserting a PDF (a binary format) into a textual prompt template and second, the length of the text was way too long. Even if we could extract the text from the PDF to insert it into the prompt, it put us way over the token limit allowed by the LLM.

We solved the first problem by using a free, open-source utility named PdfPig by UglyToad. I'll save the details of this for another article, as this is a pre-processing step and doesn't directly involve SK. Once we had the raw text, we needed to break it up into chunks small enough that we could stuff one or two of the most relevant sections of the contract into the prompt without hitting the token limit for the LLM.

We found that just blindly cutting a document into sections of a certain length didn't work very well. Imagine if we created the following chunks, "...MikeYeager.com herein kno", and "wn as CLIENT...". We would never be able to determine that the company MikeYeager.com was also referred to as CLIENT in the contract, because we cut that thought in half. Chunkers need to be smart enough to respect paragraph and sentence boundaries when deciding where to start and end each chunk, so each chunk captures complete thoughts. We also found that it helps to include a little overlap from the previous and next chunk for the same reason. It's trickier than you might think, but luckily, a lot of people have already solved this problem well and we didn't have to code this from scratch.

We solved the second problem by incorporating the TextChunker class included with Semantic Kernel. By the time SK V1 was released, the TextChunker was marked for evaluation purposes only and it looks like the team may drop it in the future. Because chunking is also a pre-processing step and doesn't directly involve SK (the primary reason we suspect it will be dropped), I'll also cover that in the future article I mentioned.

After creating a new text extraction and chunking utility, we could run a pre-processing step to save the plain text chunks of all the contracts to disk. Now, when the user asked a question, we first asked the LLM to extract search terms from the question with a prompt like this one:

```
Extract search terms suitable for a keyword
search from the following.
Separate each term with a comma.
####################
{{$input}}
####################
```

Then we wrote C# code to search the text files from a particular contract for those keywords. We took the top one or two matches and injected the text of those chunks into the prompt instead of injecting the entire contract. The RAG code to find relevant chunks of text for a contract can be found in **Listing 1**.

With all the pieces in place, the Copilot could now answer most questions, even if it was a little bit slow. For example, if the user asked, "Is the contract currently in effect?", the final prompt sent to the LLM might look like **Listing 2**. The final original implementation with all these steps can be seen in **Listing 3**.

Listing 1: RAG code to find relevant chunks of the specified contract.

```
public string FindRelevantChunks(
  string contractFileName, string keyWords)
{
  var results = new List<ChunkInfo>();

  var keywords = keyWords.Split(',');

  var justStem = Path.GetFileNameWithoutExtension(contractFileName);
  var i = 0;
  var potentialFileName = Path.Combine("SampleDocuments",
    $"{justStem}_{i.ToString().PadLeft(4,'0')}.txt");
  while (File.Exists(potentialFileName))
  {
    var fileContents = File.ReadAllText(potentialFileName);

    var keywordCount = 0;
    foreach (var keyword in keywords)
      keywordCount+= Regex.Matches(fileContents, keyword,
        RegexOptions.IgnoreCase).Count;

    results.Add(new ChunkInfo
      { Text = fileContents, KeywordCount = keywordCount });

    i++;
    potentialFileName = Path.Combine("SampleDocuments",
      $"{justStem}_{i.ToString().PadLeft(2, '0')}.txt");
}
```

An Updated Approach

While writing the original version, we thought we could use **planners** in the future to reduce the amount of calls we were making and the amount of hand coding we were doing. Planners are an SK concept that asks the LLM to create a set of steps to be executed to accomplish a complex task. Based on the user's question, the planner would examine both the native and semantic functions available to it, and come up with a set of steps, which we could inspect and even interact with. SK then executed the steps, passing the results of one step to the next. It was a pretty good idea, and we had some success with it, but our attempts to use planners at the time found them to be error prone and it was a lot of work to make relatively simple decisions. What we had hand coded worked well enough and was faster, so we went with that.

Recently, my team was given the chance to re-evaluate our solution and see if we could use new advances, such as GPT-4 and automatic function calling to improve on our Copilot. Inspired by OpenAI, automatic function calling is similar to planners, except that we found it a little less ambitious, a little more automatic, and a little less error prone. Automatic function calling allows us to load a set of function descriptions and signatures into SK (just like we did with planners) then, based on those descriptions, have SK automatically determine which functions (if any) to call and when. Automatic function calling is simpler than planners because it doesn't require coming up with the entire set of steps up front. Instead, it determines if and when to call a function as it runs, so it can adapt to new information and make simpler decisions. Then SK automatically runs the functions and incorporates the results. It sounded promising.

> We turned our original inline C# code into full-fledged native SK functions to allow GPT-4 to call them automatically.

In the OpenAI version of automatic function calling, when a function is to be called, the API initiates a callback with the name of the function to be called as well as values for the parameters it determines need to be passed to the function. When a callback happens, our application oversees making the call and returning the results to OpenAI for further processing. With SK however, the callbacks are handled automatically for us. SK invokes the functions, manages the parameters and return values, and injects the response into the prompt when and where it's appropriate. It was impressive, but it only worked marginally well with the GPT-35-TURBO model we were using. Though slower and a bit more expensive, the GPT-4 model proved to be quite good at this type of work and did a fine job of calling the right functions and generating

Listing 2: The final prompt sent to the LLM in the original Implementation.

```
The current date and time is 3/27/2024 10:8AM
Based on the following:
#############
Address: 1234 Oak St.
City: Houston
State: TX
ZIP: 77333
Apartment Number: 1235

Term of the Agreement:
The rental term will commence on 02/14/2020 and
terminate on 02/13/2021. This Agreement is valid
for a period of twelve months.

Rent Payment:
a. The monthly rent for the Apartment is set at
$799 and is payable on or before the 1st of each
month. The first payment will be due on 02/14/2020.
b. Rent payments should be made in check
payable to the Landlord at the address mentioned
above or as otherwise instructed by the Landlord.

Security Deposit:
a. The Tenant(s) shall provide a security deposit of
$500 on or before the commencement of this
Agreement.
b. The security deposit will be held by the
Landlord as security against any damage,
unpaid rent, or other charges owed by the Tenant(s)
under this Agreement.
c. The security deposit will be returned within 10
days after the termination of this Agreement,
subject to deductions for any unpaid rent or
damages as outlined in Section 5.
#############
Answer the question:
Is the contract currently in effect?
```

Listing 3: Complete original version.

```
private static async Task AskQuestionsAboutContract1()
{
  var builder = Kernel.CreateBuilder();

  builder.Services
      //.AddLogging(configure => configure.AddConsole())
      .AddAzureOpenAIChatCompletion(
          "gpt-35-turbo",
          _endpoint,
          _apiKey);

  var kernel = builder.Build();

  var semanticFunctions =
    kernel.ImportPluginFromPromptDirectory("Semantic");

  //TODO: Add UI to have user select the contract they want
  //to work with and enter questions they want to ask...
  var currentContract = "SampleContract1";
  var question = "Is the contract currently in effect?";

  var needsDateTimeResult = await
    kernel.InvokeAsync(semanticFunctions["NeedsCurrentDateAndTime"],
      new KernelArguments { ["input"] = question });

  Console.WriteLine(
    $"Needs Date and/or Time? {needsDateTimeResult}");

  var dateTime = string.Compare(needsDateTimeResult.ToString(),
    "YES") == 0
      ? $"The current date and time is {DateTime.Now}"
      : string.Empty;

  Console.WriteLine($"dateTime parameter: {dateTime}");

  var searchTerms = await
    kernel.InvokeAsync(semanticFunctions["ExtractSearchTerms"],
      new KernelArguments { ["input"] = question });

  Console.WriteLine($"Search Terms: {searchTerms}");

  var chunkSearcher = new SearchChunks();
  var relevantChunks =
    chunkSearcher.FindRelevantChunks(currentContract,
    searchTerms.ToString());

  Console.WriteLine($"Relevant Chunks: {relevantChunks}");

  var result = await
    kernel.InvokeAsync(semanticFunctions["SearchContract"],
      new KernelArguments
      {
          ["input"] = question,
          ["dateandtime"] = dateTime,
          ["contract"] = relevantChunks
      });

  Console.WriteLine();
  Console.WriteLine($"Working with contract: {currentContract}");
  Console.WriteLine(question);
  Console.WriteLine(result);
}
```

Listing 4: Calling functions automatically.

```
private static async Task CallFunctionsAutomatically()
{
  var builder = Kernel.CreateBuilder();

  builder.Services
    .AddLogging(configure => configure.AddConsole())
    .AddAzureOpenAIChatCompletion(
    "gpt-4",
    _endpoint,
    _apiKey);

  var kernel = builder.Build();

  var nativeFunctions = kernel
    .ImportPluginFromType<DateAndTime>();

  OpenAIPromptExecutionSettings settings = new()
  {
      ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions
  };

  var result = await kernel.InvokePromptAsync(
     "What is today's date?", new(settings));

  Console.WriteLine(result);
}
```

a correct answer almost all the time. Even though the model is slower, it was making fewer calls on our behalf than our original solution, so overall, it was faster.

Our first task was to turn our original inline C# code into full-fledged native SK functions to allow GPT-4 to call them automatically. All we had to do was add the KernelFunction and Description attributes to our existing methods. No other changes were required.

```
[KernelFunction,
  Description("Returns the current date and time")]
public DateTime GetNow()
{
  return DateTime.Now;
}
```

Listing 4 shows a simple example of automatic function calling in a console app with logging enabled so you can see what's going on under the hood. After loading a set of native and/or semantic functions into the kernel, we run the prompt, passing a setting of Auto**InvokeKernelFunctions = true,** indicating that that SK can call any of the loaded functions, or any of the out of the box functions provided by SK on our behalf, as it sees fit. In this case, we've loaded the GetNow native function into the kernel, which has the following description: "Returns the current date and time". If we examine the logs when we ask the question, "What is today's date?", we'll see that SK opts to call our GetNow function and automatically includes the response in the prompt, without any intervention from us, before asking the user's question to the LLM. Provided with this information in the prompt, the LLM then correctly answers the question.

If we comment the ToolCallBehavior setting, we'll see that the LLM can no longer answer the question because it can no longer call our functions or the out of the box functions that come with SK. Similarly, if we uncomment the setting and change the question to "What is the capital of New Mexico?", we'll see in the logs that SK no longer calls the GetNow function, because the information is no longer relevant in answering the question. The current date and time are such common pieces of information that SK actually includes them in the out of the box implementations we get for free, but we show it here as a simple, but powerful illustration.

We soon realized that we no longer needed most of the semantic functions (prompts and settings) that we'd created for our original version. GPT-4 was good enough to figure out whether it needed to call our GetNow function, to generate search terms, to call FindRelevantChunks, to pass the right parameters, and to inject the results into a prompt it generated on the fly without any help from us! With a little tweaking, using only the GetNow and FindRelevantChunks native functions, we could now turn on the AutoInvokeKernelFunctions setting and ask the

Listing 5: Updated version using automatic function calling and GPT-4

```
private static async Task AskQuestionsAboutContract2()
{
  var builder = Kernel.CreateBuilder();

  builder.Services
    //.AddLogging(configure => configure.AddConsole())
    .AddAzureOpenAIChatCompletion(
      "gpt-4",
      _endpoint,
      _apiKey);

  var kernel = builder.Build();

  kernel.ImportPluginFromType<DateAndTime>();
  kernel.ImportPluginFromType<SearchChunks>();

  OpenAIPromptExecutionSettings settings = new()
  {
    ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions
  };

  var currentContract = "SampleContract1";
  Console.WriteLine($"Working with contract: {currentContract}");
  var question = "Is the contract currently in effect?";
  await AskQuestion(kernel, settings, currentContract, question);

  question = "What jusrisdiction governs the contract?";
  await AskQuestion(kernel, settings, currentContract, question);

  question = "Who pays for electricity?";
  await AskQuestion(kernel, settings, currentContract, question);

  currentContract = "SampleContract2";
  Console.WriteLine($"Working with contract: {currentContract}");

  question =
    "How long does the depositor have to deposit materials?";
  await AskQuestion(kernel, settings, currentContract, question);

  question = "What is Tower48's maximum liability?";
  await AskQuestion(kernel, settings, currentContract, question);
}

private static async Task AskQuestion(Kernel kernel,
  OpenAIPromptExecutionSettings settings, string currentContract,
  string question)
{
  Console.WriteLine(question);
  var result = await kernel.InvokePromptAsync(
    $"ContractFileName: {currentContract}. Question: {question}",
    new(settings));
  Console.WriteLine(result);
  Console.WriteLine();
}
```

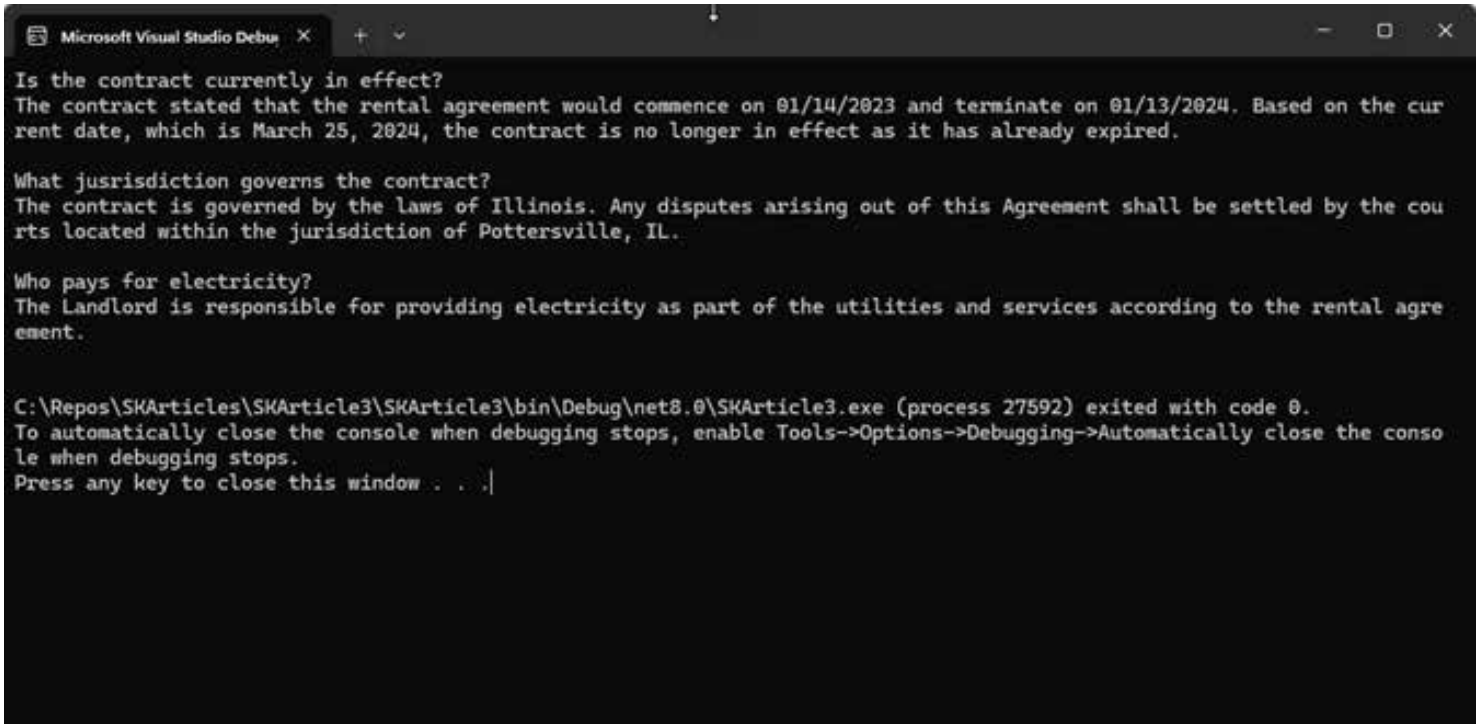

Figure 1: Test questions and results about an apartment rental agreement

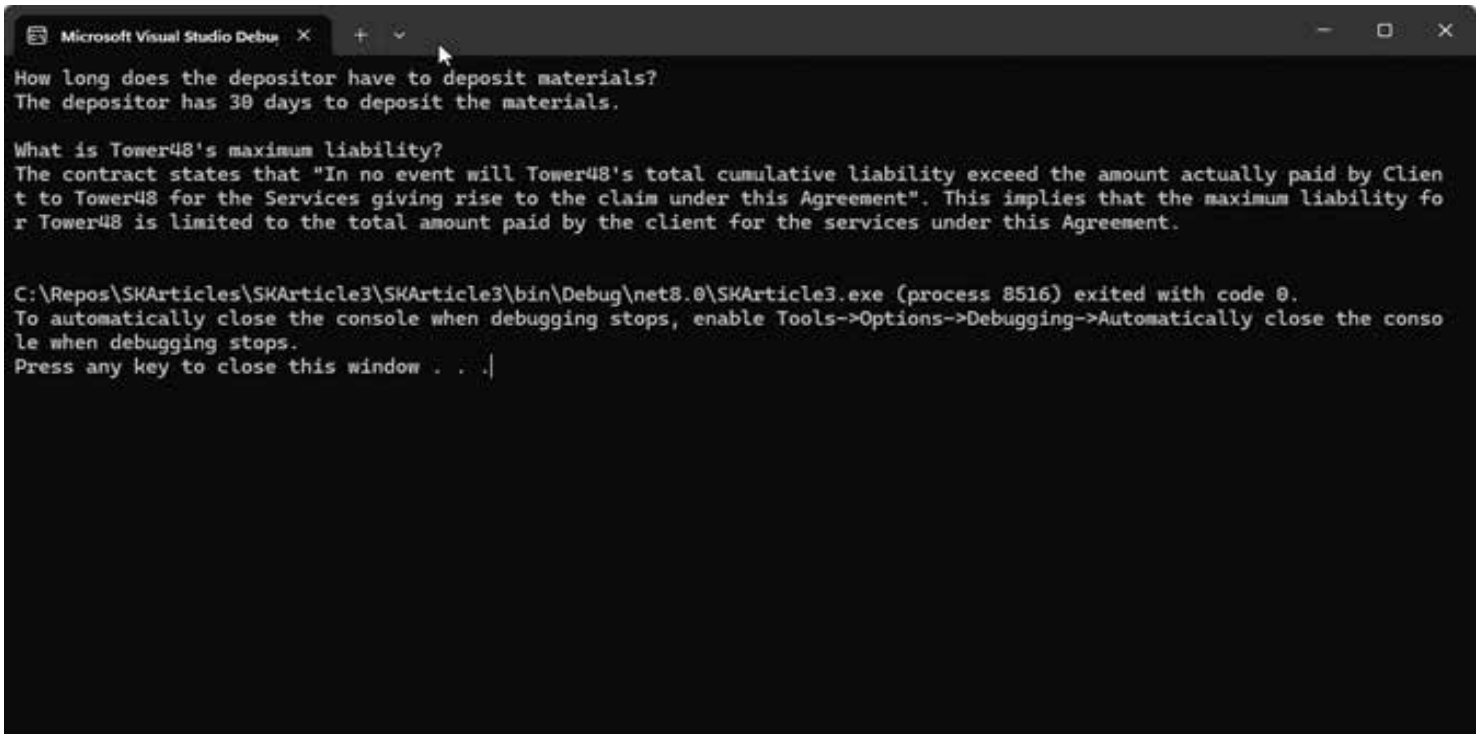

Figure 2: Test questions and results about an escrow contract for digital assets

LLM to answer almost any user question. SK and GPT-4 now handled almost everything else. **Listing 5** shows the updated code.

As an example, **Figure 1** shows the series of test questions posed against the first sample contract, an apartment rental agreement.

Figure 2 shows questions posed against the second contract, an escrow contract for digital assets.

Summary

In our original implementation, we did a lot of work by hand to get the GPT-35 and GPT-35-TURBO models to do

their magic. We had to determine the user's intent, we had to figure out what information was needed to retrieve the answer to the question, we had to write functions to retrieve the information, we had to inject the information into the prompt, and we had to engineer all our prompts to get good results, all so the LLM could generate a friendly and correct response. After GPT-4 and automatic function calling, we got rid of the lion's share of code, and we now get even better results with extremely basic prompts!

> We now get even better results with extremely basic prompts

Looking back on this experience several projects later, the lessons learned building our first Copilot were fundamental and we've used them in nearly every project since. Even though we're continually improving our approach to building Copilots and various other flavors of AI and even as the LLMs and tools we use are getting better and better, those early lessons are still relevant, important, and useful. In the next article, I'll take a look at more advanced RAG implementations and the lessons learned creating them.

Mike Yeager

CODE 30 YEARS

Sep/Oct 2024
Volume 25 Issue 5

Group Publisher
Markus Egger

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers in This Issue
Markus Egger
Joydip Kanjilal
Wei-Meng Lee
Nevio Medancic
Mike Yeager
Enzo Grubisa
Vassili Kaplan
Sahil Malik
Paul D. Sheriff

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@code-magazine.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: Ingram Periodicals, Inc.
International Bonded Couriers (IBC)
Media Solutions
Source Interlink International

Subscriptions

Circulation Manager
Colleen Cade
832-717-4445 ext. 28
ccade@codemag.com

US subscriptions are $29.99 USD for one year. Subscriptions outside the US are $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa and Discover credit cards accepted. Back issues are available. For subscription information, email subscriptions@code-magazine.com or contact customer service at 832-717-4445 ext. 9.

Subscribe online at
www.code-magazine.com

CODE Developer Magazine
EPS Software Corporation / Publishing Division
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379 USA
Phone: 832-717-4445

CODE
STAFFING

UNLOCK STAFFING EXCELLENCE

Top-Notch IT Talent, Contract Flexibility, Happy Teams, and a Commitment to Customer Success Converge with CODE Staffing

Our IT staffing solutions are engineered to drive your business forward while saving you time and money. Say goodbye to excessive overhead costs and lengthy recruitment efforts. With CODE Staffing, you'll benefit from contract flexibility that caters to both project-based and permanent placements. We optimize your workforce strategy, ensuring a perfect fit for every role and helping you achieve continued operational excellence.

Ready to Discuss Your IT Staffing Needs?

Visit our website to find out more about how we are changing the staffing industry.

Website: codestaffing.com
Yair Alan Griver (yag)
Chief Executive Officer
Direct: +1 425 301 1590
Email: yag@codestaffing.com