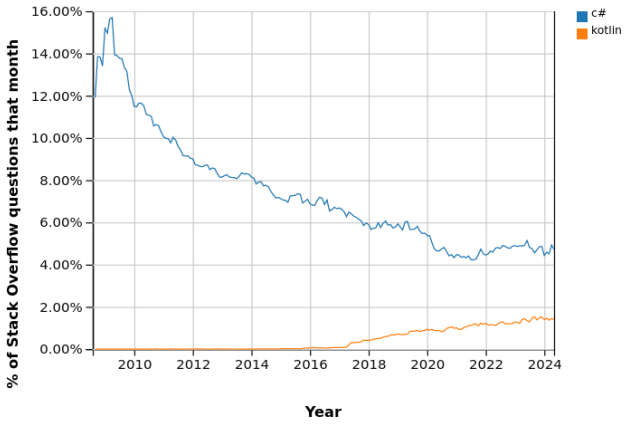
A Comparative Analysis of C# and Kotlin:

A Tale of Two Programming Languages by Glenn Engstrand

When it comes to developing high-performance microservices, choosing the right programming language and ecosystem can make all the difference. In this article, we'll compare two popular options: .NET/C# and JVM/Kotlin. We will cover the developer experience of and performance under load for these two very different, yet also eerily similar,



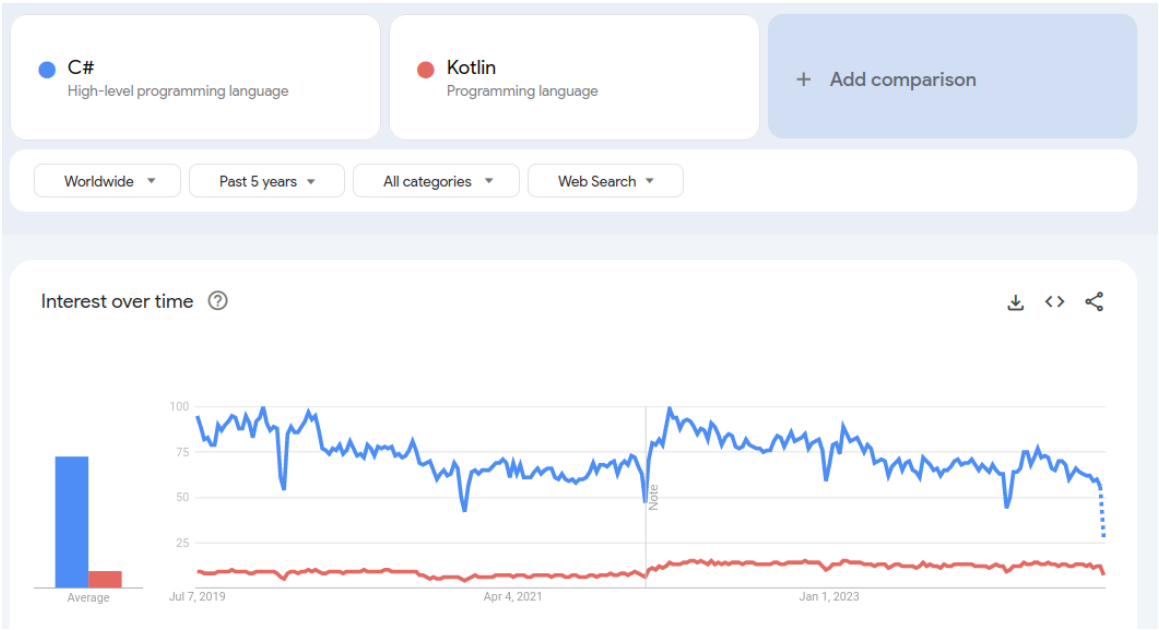
programming languages and tech stacks as they relate to business focused microservice development. This is also a story about the different approaches that companies can take when promoting the use of open source that they develop and share with the rest of the world.

Historical Context

But first a little back-story to set the context. When I want to learn more about a particular programming language or tech stack, I have this github repository where I develop multiple open source implementations of a feature identical, polyglot persistent, rudimentary news feed microservice. In that repo, you will also find the code and configuration for a load test lab by which I subject each implementation to then record the performance based analytics for the purposes of making comparisons between the different implementations.

The previous blogs here covered the first 13 implementations in that repo which used various open source technologies. This blog covers implementations of feed 14 and feed 15 in that repo which uses more corporate open source technologies. You may be wondering what the difference is. After all, isn't most widely supported successful open source projects backed by corporations with deep pockets these days? Perhaps this is somewhat subjective but it feels to me that companies like Oracle, Google,

Lightbend, and Nubank keep their respective general purpose open source programming languages and related technologies at arms length through some kind of authentic community process which provides oversight into the direction that Java, Golang, Scala, and Clojure takes. This blog is about Microsoft and JetBrains which keep their technologies, .NET/C# and Kotlin respectively, close and under their tight control.



This is pretty much where the similarities between the two companies end. Founded in 1975, Microsoft is headquartered in Redmond, WA and has approximately 221,000 employees. Founded in 2000, JetBrains is headquartered in Prague, Czech Republic and has around 1,800 employees. Some folks may believe that Kotlin comes from Google since Google promotes the use of Kotlin for Android development. This blog focuses on Kotlin vs C# from the perspective of backend server-side microservice development.

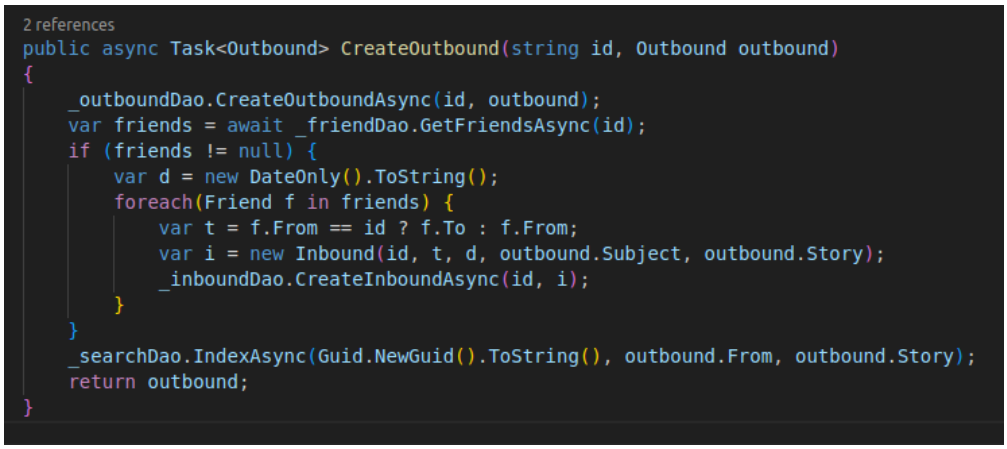
Feed 14 (Microsoft based C# on ASP.NET)

The tech stack is C# running in the .NET CLR (Common Language Runtime) hosted in the ASP.NET web server framework. This is version 7. My first exposure to .NET was version 1. That was back in the days when Microsoft was run by Steve Ballmer. He was all about bundling. If you wanted to run .NET, then you had to run it on Microsoft Windows. If you wanted to build

and run a web service, then it had to be on IIS (Internet Information Server). Satya Nadella is the current CEO of Microsoft and he has a different strategy. Modern .NET can easily and naturally be run in Docker on Linux powered Kubernetes. Web services can be run on the cross platform Kestrel which is very modern, efficient and can accommodate high performance.

There is a full featured Object Relational Mapping framework available called the Entity Framework. ORMs tend to be slower so I chose to simply go with the MySql driver for ADO.NET instead.

The IDE Experience



```
2 references
public async Task<Outbound> CreateOutbound(string id, Outbound outbound)
{
    _outboundDao.CreateOutboundAsync(id, outbound);
    var friends = await _friendDao.GetFriendsAsync(id);
    if (friends != null) {
        var d = new DateOnly().ToString();
        foreach(Friend f in friends) {
            var t = f.From == id ? f.To : f.From;
            var i = new Inbound(id, t, d, outbound.Subject, outbound.Story);
            _inboundDao.CreateInboundAsync(id, i);
        }
    }
    _searchDao.IndexAsync(Guid.NewGuid().ToString(), outbound.From, outbound.Story);
    return outbound;
}
```

I did all of the .NET coding using Visual Studio Code. This is an open source electron app code editor under the MIT license. This is not the same as Visual Studio which is a full featured IDE that, you know, has a very enterprise targeted pricing structure and is more focused these days on cross selling into Azure. VS Code has a great developer experience. It doesn't feel like a feature locked or dummied down version at all to me.

The Build Chain

The CLI build tool is called dotnet. Unlike the more popular JVM targeting build tools such as Maven, SBT, or Gradle, it doesn't have a plugin architecture. You can't integrate your own extensions to the build chain. It supports most of the operations that you come to expect such as new, build, run, test, publish, and format (linter). The open source, freely available repository of build artifacts from which the dotnet build tool can download dependencies is called nuget.

There is a code coverage capability but I could not get it to work on the free

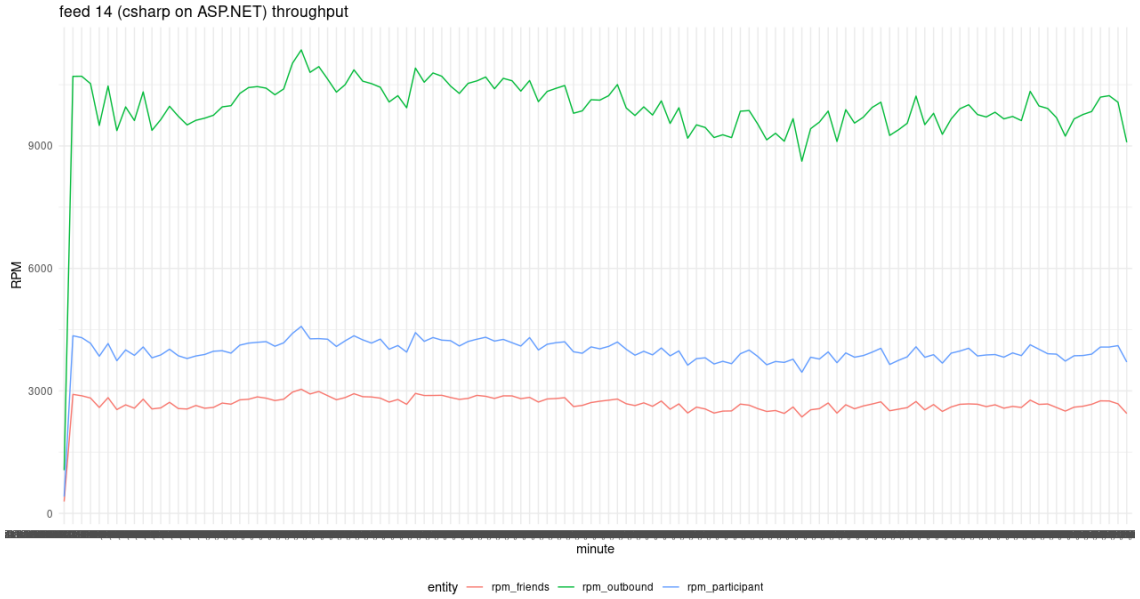
version. There are 3rd party commercial offerings for both code coverage and vulnerability scanning. The dotnet CLI doesn't directly support the building of docker images or generators such as grpc or openapi. This is not a big problem as there are plenty of other tools that provide all of that capability to .NET developers.

I could not find a free profiler for modern .NET that I felt comfortable installing on my local laptop. Curiously, the best commercial profiler for .NET comes from JetBrains which is bundled into their enterprise upsell of their IDE for .NET called Rider.

Here is another curious difference. Test automation assets are not embedded within the service under test. You have to put that in a separate project with its own build file. This is not a big problem as you can put both folders under the same repo.

Performance Results

As mentioned earlier, I run a multi-threaded load test where participants are created, friend each other, then publish outbound events. Everything is running in GKS (Kubernetes on Google Cloud). Average RPM for the add participant operation is 3970 for the add friend operation is 2677 and for the add outbound operation is 9919. Average duration for the add outbound operation is 11 ms and the 95th percentile is 16 ms. These are very respectable numbers.



Static Code Analysis

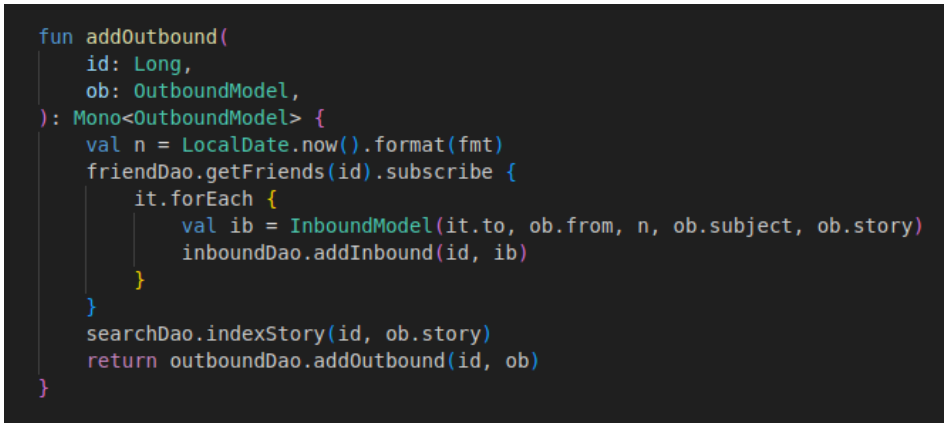
The C# implementation of the news feed service totaled 1037 lines of code within 39 files. That's pretty good in terms of striking a healthy balance between expressivity and terseness. Comparing with all of the programming languages from all of the feed implementations by average lines of code per file where lower is better, this puts C# at the top of the list and Kotlin somewhere in the middle.

Feed 15 (JetBrains based Kotlin on Broadcom based Spring Webflux)

The tech stack is Kotlin built by Gradle 8 for JDK 17. The web framework is Webflux which is the reactive extension for Spring boot (3.2). Relational DB access is via Spring R2DBC. Whereas there is only one CLR which comes from Microsoft, there are JDKs from at least a dozen companies. The version of the JDK used here is Temurin which is released by Adoptium which is a top level project under the Eclipse Foundation which was founded by IBM but now has hundreds of member companies. Corretto from Amazon is another popular choice.

The IDE Experience

Like the .NET code, I also coded this Kotlin service using Visual Studio Code. As already mentioned, Kotlin was invented by a company called JetBrains which is



```
fun addOutbound(
    id: Long,
    ob: OutboundModel,
): Mono<OutboundModel> {
    val n = LocalDate.now().format(fmt)
    friendDao.getFriends(id).subscribe {
        it.forEach {
            val ib = InboundModel(it.to, ob.from, n, ob.subject, ob.story)
            inboundDao.addInbound(id, ib)
        }
    }
    searchDao.indexStory(id, ob.story)
    return outboundDao.addOutbound(id, ob)
}
```

the company behind another editor called IntelliJ. I suspect that this competing code editor is their main source of revenue (but they do have like thirty different products that they sell) and they release the Kotlin compiler yet I had a very smooth and functional experience with VS Code. Not only does JetBrains not attempt to sabotage the developer experience for Kotlin on VS Code, but the IntelliJ IDEA CE (community edition a.k.a. free) is also

pretty good. Like VS Code, IntelliJ IDEA CE doesn't feel like a feature locked or dummied down version at all to me. The download link for the CE is a little hard to find on their website.

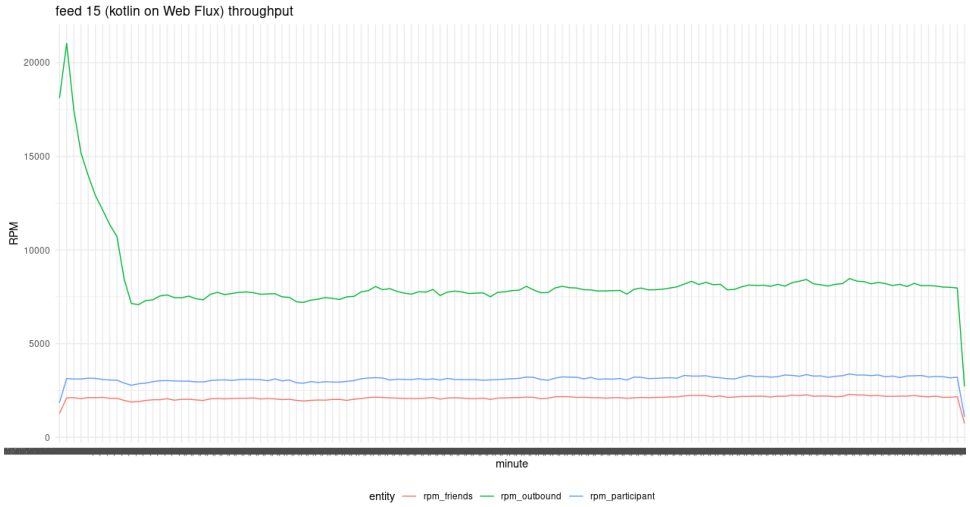
The Build Chain

Gradle is the preferred build and dependency management tool for Kotlin which is what I used here. It has a plugin architecture with a rich ecosystem of extensions. Without any issues, I was able to use ktlint for linting and formatting, JUnit for test automation, Jacoco for code coverage, and JDK Mission Control for profiling. The open source, freely available repository of build artifacts from which the Gradle build tool can download dependencies is called Maven Central.

Performance Results

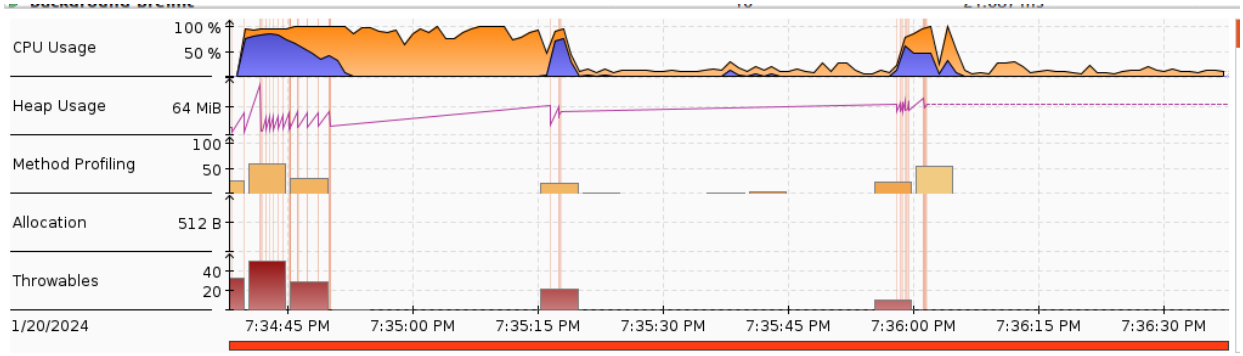
I ran the Kotlin implementation with the same load test and the environment as the C# implementation. Average RPM for the add participant operation is 3107 for the add friend operation is 2097 and for the add outbound operation is 8299. That is 17% less throughput than the C# version. Average duration for the add outbound operation is 13 ms and the 95th percentile is 110 ms.

I am not entirely sure why the performance of the Kotlin service was poorer than the C# service. If I had to guess, then I would suggest that it was possibly because of the Spring components. While not as heavy weight an ORM as Hibernate from Spring Data JPA, Spring R2DBC is still less efficient than just the straight up MySql ADO.NET driver.



Static Code Analysis

This service has 37 Kotlin files totaling 2037 lines of code. That is almost twice the size of the C# implementation. What happened? When you reference a type in a program from a different package, you have to tell the compiler which package that type can be found in. You use the `using` keyword in C# to do that and the `import` keyword in Kotlin. The Kotlin linter forces you to import each type in a separate line of code. In the C# code, I would just specify the entire package (including all types in that package) in a single using statement. Because of that difference, there are three times as many import statements in the Kotlin code as using statements in the C# code.



Language Features

What is similar with both C# and Kotlin is that they are both modern programming languages that support many emerging popular features. Examples include null safety, data classes, type inference, extension functions, properties, and dual paradigm object oriented and functional programming.

Obviously, there are differences in the syntax of the two programming languages but I am not interested in that level of minutiae here. Perhaps how the two programming languages handle concurrency might be worth mentioning. C# takes a page from javascript with the async and await keywords in a threading construct known as a task. Since Kotlin runs on the JVM with a close interoperability with the JDK, it can use any kind of concurrency available for Java. Kotlin adds first class support for suspendable

coroutines (i.e. structured concurrency) on top of all that. You won't see any of that in feed 15 due to the tight integration with Webflux and Spring R2DBC which both depend heavily on the mono publisher from project reactor for concurrency.

A Metaphorical Comparison

I started this blog by lumping both C# and Kotlin into the same corporate open source category but (borrowing from the metaphor presented in Eric Raymond's 1997 essay entitled The Cathedral and the Bazaar) C# is more like the cathedral while Kotlin is more like the bazaar. JetBrains controls the Kotlin language and compiler but not the runtime that it operates in nor any other aspect of the developer ecosystem (except for their code editor which is not mandatory). Microsoft controls almost all aspects of C# including its runtime and ecosystem. The source code for .NET/C# may be under the MIT license and there is a non-profit .NET foundation but it is crystal clear to me whose house I am a guest in. Earlier in this blog, I mentioned that Maven is the repository for Kotlin and Nuget is the repository for C#. Maven has almost twice as many packages as Nuget.

Worldwide, Jul 2024 :

Rank	Change	Language	Share	1-year trend
1		Python	29.35 %	+1.5 %
2		Java	15.6 %	-0.2 %
3		JavaScript	8.49 %	-0.8 %
4		C#	6.9 %	+0.1 %
5		C/C++	6.37 %	-0.1 %
6	↑	R	4.73 %	+0.3 %
7	↓	PHP	4.49 %	-0.5 %
8		TypeScript	2.96 %	-0.1 %
9		Swift	2.78 %	+0.2 %
10	↑	Rust	2.55 %	+0.4 %
11	↓	Objective-C	2.39 %	+0.2 %
12		Go	2.12 %	+0.2 %
13		Kotlin	1.91 %	+0.1 %
14		Matlab	1.53 %	-0.1 %

Conclusion

If you like the consistency, uniformity, performance and perhaps better operability of the cathedral, then C# is most probably for you. If you prefer a larger number of more diverse options and choices offering a wider degree of freedom of coding expression, then consider Kotlin.