# Star 2-1-2022 National General Insurance

Situation:

Business unit reporting delay in updating producer license data.

## Task:

A backend service which processes data from a 3rd party Api started taking days to complete. Normally it takes hours.

## Action:

As the owner of this process I setout to try and figure out what caused it to slow down.

1st I started looking at the logs which I could see the process was taking 36-48 hours to finish. I then started looking at older logs hoping to see when this started. Apparently, it started about sometime in Feb 2022. Logs for the prior month revealed the normal processing time of 6-8 hours. I now knew WHEN the issue started.

2nd Now that I knew when the slowdown occurred, my next step was to figure out WHY. For this I had to ask the question “what changed”? Fortunately, we keep detailed release notes and all our code changes are tied to a work item in Azure DevOps. So, I started looking at everything we released during the month when the slowdown started. Nothing really stood out. No smoking guns.

3rd With no obvious code issue sticking out it was back to troubleshooting 101. In an attempt to eliminate any infrastructure changes which might have caused this I ran the application in a test environment. While running the application I was watching the logs and over time the amount of time it took to process an individual producer was steadily increasing. This indicated that the process was slowing down as it ran. I then re-started the process, but this time I had the Windows task manager open to watch the process as well as the logs. Now I could see the memory usage going up, up, and up. It never leveled out. Eventually it took all the available memory on the VM! I asked the infrastructure people to double the RAM from 8gb to 16gb and re-started the process. As I suspected it took all the extra memory as well. At this point I was confident I had a memory leak. The question remained “what was causing the memory leak”.

4th Now that I knew I had a memory leak, and that leak did NOT occur prior to the Feb 2022 release I knew some code change caused this. But what? Since this process runs on a Windows VM I was able to run it in the test environment and create a memory dump while it was running. This would give me a runtime snapshot of what was going on internally. Well it turns out our corporate version of Visual Studio did not have the feature which allows you to inspect a memory dump. I took it upon myself to install JetBrains dotMemory (the .Net memory profiler). Armed with this tool I could now see the objects in my application as they existed while running. There were approx. 92 million CLR objects in this 6.8gb memory dump file. From this I could clearly see millions of our EF Core DB contexts were not getting disposed. WHY? Everything was properly wrapped in a C# using statement. What changed?

5th “What would cause all of our EF Core DB contexts to not dispose themselves”? To answer this question, I had to go back into our Git history to see the code commits for the Feb 2022 timeframe. I wanted to see if any of them changed. SURE, ENOUGH THEY DID! While upgrading the application from .Net 3.1 to .Net 5 one developer re-factored all the EF Core DB contexts. Moving their creation inside the DI container. I now knew we had an issue disposing EF Core contexts in Net Core 5! It is at this point I started Googling for an answer.

6th All our EF Core DB contexts were now registered as “Transient” in the DI container. This was a change from the original code where we created the contexts manually in the code. Now we were passing the creation job to the .Net DI container. The smoking gun and the solution came to me as I was scanning a EFCore GitHub issue entry. *“When you register a transient service which implements IDisposable, by default the container WILL HOLD onto these references and NOT dispose of them until the container is disposed when the application stops”* By moving these contexts into our DI container and registering them as transient we inadvertently prevented them from ever getting disposed by the garbage collector.

Result:

Now that I knew what was going on all we had to do was to revert the changes specific to the EF Core DB context refactor. I reverted the changes and re-deployed the code to our test environment. Sure enough the memory leak went away.

# Why am I

I want to get back to building products. For the first 3 years I worked on PLL (Producer License Library). This was a high visibility project run by a well respected senior architect. Joining his team was exciting as I was learning and building alongside on of the top performers at the company! The experience was incredible. As the project wined down the architect passed the ownership of this product to me and moved on. Today I am this product’s team lead, managing 2 full time developers, as we continue to support and enhance this product.

Recently my group “Enterprise Services” took control of a very old suite of Api’s and batch processes called “vendor services”. This code was written over 10 years ago in an outdated language by outsourced development teams. Since I had great success with PLL leadership determined that I would be perfect to take on these new responsibilities as well. I had no idea what I was in for. Since taking over these legacy services my team spends about 50% of their time just keeping these services operational! Some of the daily tasks we now much take on include:

* + Manually updating data in test environment to support other teams testing efforts.
  + Manually updating data in production environments. There are applications which we do not own or control which may corrupt the data in a database we now own. When this occurs, we must manually fix the data.
  + First point of contact if some service does not work as expected.
  + As there are over a dozen applications calling these services from various parts of the organization, we are now required to scan several e-mail distribution lists for signs of trouble with these services.
  + We do not know all the application which use these services as cross company communication is very much lacking.
  + Very much in the dark about decisions which affect my team. I am not able to accurately forecast or plan as I have no awareness of upcoming projects or initiatives. I find out things that directly affect my team second hand, like a RabbitMQ implementation or a cloud migration that affected our services.

The first 3 years were great as we built a killer application suite along with a great team. Then in early 2022 management decided to assign my team a legacy service layer which was