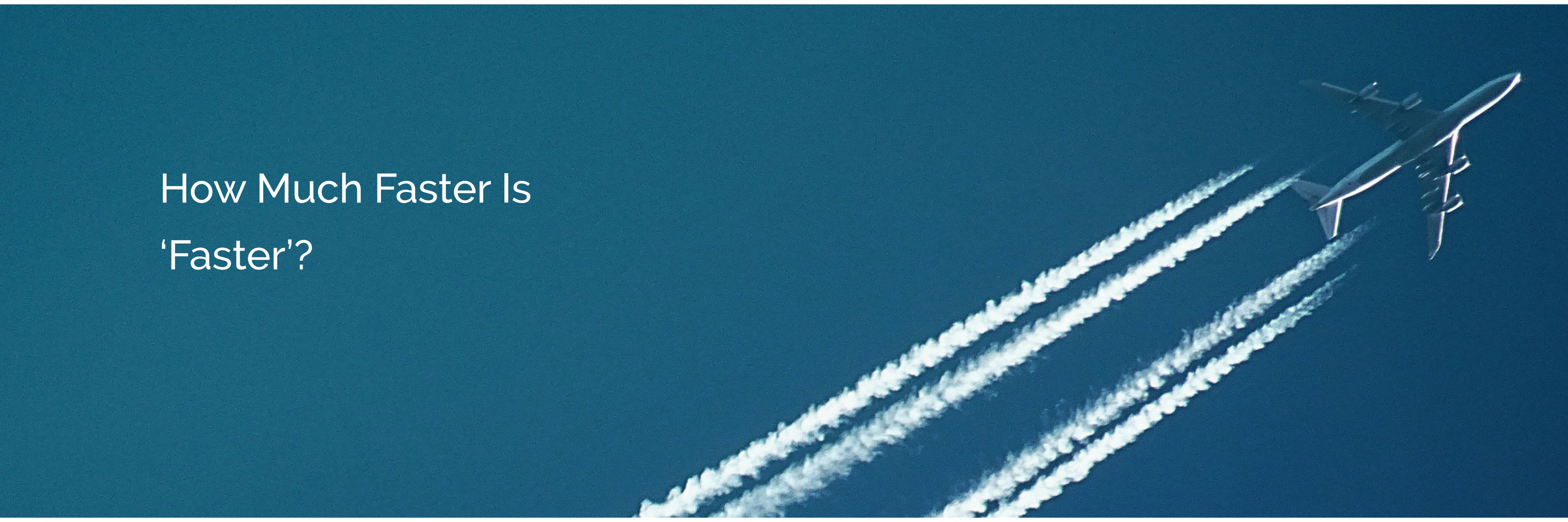
**CASE STUDY:**

# Legacy Application Confirmation Study



At Monza Cloud, we know the capabilities of AzStudio software to move you into Azure faster, through our own development efforts and in seeing how our customers use the software. However, to increase our understanding, we really wanted to test AzStudio's speed-to-delivery for a very common use case—updating legacy applications. No gimmicks. No unfair advantages. Simply an objective exhibition to substantiate the power and the advantage of using the AzStudio toolset vs. going it alone.
Ultimately, we wanted to be able to quantitatively answer the question, "How much faster is developing with AzStudio than programming with only the raw Azure application programming interfaces?"

## The Hypothesis

In order to showcase AzStudio's unprecedented versatility and efficacy, our team prepared to replicate a side-by-side performance comparison of a legacy app modernization initiative—where AzStudio development tools were pitted against a straight conversion to Azure PaaS using only Azure's APIs.  Based on the results we typically see with AzStudio, our developers predicted that, even in the face of an unknown, untested legacy application, AzStudio would deliver a faster solution than manual processes and would provide an infrastructure capable of supporting more features within the app itself.

The team decided to use a legacy Twitter application—an app that no one on the development team was familiar with—for their test case as it was a good example of a sensible on premises legacy .NET application that was not likely to be sunsetted or too expensive to rule it ineligible for migration. The modernized legacy application would be much cheaper and better as a native PaaS app.

At the end of the experiment, our developers would compare the updated applications produced by the different modernization efforts for time to value and quality of the final application.

## The Experiment

In order to standardize the expertise and manpower used throughout the experimentation process, we chose to use a beginner-intermediate level developer with 2.5 years of professional programming experience in C# and .NET to conduct the performance comparisons. In fact, like many junior developers tasked with modernization efforts, the developer had very little experience using the AzStudio framework or coding against raw Azure APIs.
As you will see, some steps are repeated for both tests (setting up a Twitter account, etc.). This is important for a fair assessment, and is to be expected since we are attempting to perform the same migration via different methods.

### WHAT IT TAKES: AZURE API – MANUAL MODERNIZATION

The developer immediately set to work modernizing the legacy GitHub application, first by leveraging the application programming interface intrinsic to Azure PaaS. Their modernization process and time intervals are as follows:

### SETUP AND RESEARCH: 5 HOURS (300 MINUTES)

#### SETTING UP

- A Twitter account to get the necessary credentials to make search requests
- The VM on Azure for hosting

#### RESEARCHING/READING

- How to setup and use a storage container and queue
- Web roles/workers and types of storage containers to think about the best way to set up and deploy the application
- SQL versus table storage
- How to use a portal to create a SQL server with the lowest requirements necessary
- How to allow their IP address in the firewall

## CODING: 8 HOURS, 20 MINUTES (500 MINUTES)

### THE DEVELOPER

- Applied Twitter account credentials to configuration file to run client as is and see what it did
- Changed existing client from using a message queue to a cloud queue
- Created web form for searching/saving to queue
- Created worker to stream search results to queue and to read from queue
- **Used portal to**:
  - Set up server/SQL database
  - Add IP address to firewall exception
  - Create status and user tables in SSMS
  - Add connection info to App.config
- Wrote stored procedures to insert into status/user tables
- Launched packaging/deploying workers to Azure

## TESTING: 8 HOURS, 10 MINUTES (490 MINUTES)

### THE MONZA CLOUD DEVELOPER PERFORMED FINALS TESTS BY RUNNING, MODIFYING, AND/OR TESTING

- Client as-is to see how it performed
- Code update to cloud queue to make sure it successfully wrote to cloud queue
- Web form for searching
- Worker streaming search results to queue
- Worker reading from queue/modifying code to use cloud queue attributes (like looping)
- Data store method and preventing bad insertions using SQL commands
- Stored procedures inserting into status/user tables
- Deployed workers to make sure that they were reading/writing from the queue as well as writing SQL tables

**TOTAL TIME REQUIRED: 22 HOURS, 20 MINUTES (1340 MINUTES)**

## Manual Modernization Continued..

Once tallied up, the time it took for a single dev to complete the modernization process using Azure PaaS API was a grueling three day deep-dive (22 hours and 20 minutes). Admittedly, when a developer is tasked with the modernization of an application that they don't know, they are faced with a myriad of feature choices. This feeling is especially acute for new users to PaaS have a huge learning curve to overcome. As such, there was a great deal of testing involved to arrive at a plausible final product, and the dev's first attempt was, naturally, not perfectly efficient. Inefficiency during these types of modernization efforts is expected because most companies make junior programmers responsible for modernizing suites of legacy apps, and each modernization effort has its own unique challenges.

The developer spent quite a lot of time doing research and reading about Twitter credential security, Azure queues, and database architectures before he settled on a method that he could be confident in deploying. But, because of this planning and research, when it came time to program, the developer was able to execute his plan directly in code. He spent a bit of time tweaking and setting up database and networking features in the Azure portal. And, while his path was straightforward, he had to write a lot of code to reestablish many of the fundamental functions of that app in a PaaS-enabled environment. This, of course, also greatly improved the application vs. the original legacy version as you'll see in the results. Testing was also an elongated process, involving some trial and error, as the developer had to develop and execute tests for common issues and use a variety of dense and technical logs to debug.

## What it Takes: AzStudio Development Software – Guided Modernization

After completing the Azure API modernization challenge, the Monza Cloud developer prepared to modernize the application again, but this time, leveraging the streamlined capabilities offered through the AzStudio suite. This second modernization process and timeline breakdown are as follows. Again, you will see that we duplicated some of the effort (creating Twitter accounts, etc.) to emulate the full scope of the modernization.

### SETUP AND RESEARCH: 20 MINUTES

#### SETTING UP

- A Twitter account to get the necessary credentials to make search requests

#### RESEARCHING/READING

- Additional setup and research are unnecessary due to AzStudio's preset framework and built-in features

### CODING: 4 HOURS, 10 MINUTES (250 MINUTES)

#### THE DEVELOPER

- Created  a queue and blob for worker from existing AzStudio framework
- Created worked for stream search and queue read
- Changed existing code to use Az.QueueManager
- Wrote tables in SSMS (required some repetition because of the misreading of a stubborn column name)
- Created script and execute to create stored procedures data class
- Wrote methods to insert to tables using stored procedures
- Created plugins for status and user tables

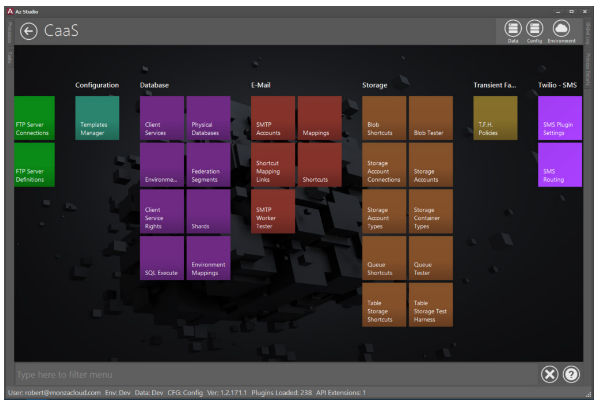
## TESTING: 3 HOURS (180 MINUTES)

### THE MONZA CLOUD DEVELOPER PERFORMED FINALS TESTS BY RUNNING, MODIFYING, AND/OR TESTING

- Stream search to queue worker
- Queue reader
- Stored procedures

**TOTAL TIME REQUIRED: 7 HOURS, 30 MINUTES (450 MINUTES)**

What took almost three full workdays to modernize on Azure APIs alone took the same developer less than a single day (7 hours and 30 minutes) to complete using AzStudio developer tools.  And, as you'll see later in the results, the new application was much more functional as a PaaS app (even more so because of some additional AzStudio functionality). The developer was able to save considerable time and effort by not having to do exhaustive research before even beginning the coding process because he was able to take advantage of the core queue and database functionality that already existed inside of the framework. And, when it came time to program, much of the foundational code was already available to the developer as part of AzStudio's toolset, so the amount of custom programming was reduced to merely customizing existing frameworks. Again, testing was improved by the guided nature of AzStudio. The Developer was able to quickly use internal tools. Like many junior programmers, our developer made a few programming errors, but by using the enhanced logging to test that the application was performing correctly, he determined that he needed to use a different type of object storage for his messaging queues and quickly resolve the issue. AzStudio's preset framework and intuitive feature selections allowed for a more user-friendly and time-efficient procedure.



User: robert@monzacloud.com Env: Dev Data: Dev CFG: Config Ver: 1.2.171.1 Plugins Loaded: 238 API Extensions: 1

## The Results – Application Quality

**The manually coded application** functioned mostly the same as the original Twitter application, with simplistic logging, error-handling, and transient fault-handling. The application, as with the original, still needed to be hardcoded with the Twitter handle to be followed. All in all, the developer had moved the application from an on-premise environment with local storage and MSMQ, to an Azure PaaS/IaaS environment with virtual storage and stream search and worker reader queues. The developer was also able to add additional SQL database tables and hand-written stored procedures to read and write to them. The modernization effort was a success! The new PaaS application became cheaper to run, easier to maintain, and now includes a built-in ability to scale capacity.

In the second phase of the experiment, the developer made all of the application changes and modifications listed above in the manual modernization, but these changes were enhanced by the AzStudio framework. This second application only partially functioned like the original legacy app—instead, it performed even better than the original. The configuration and management were changed and no VM was necessary as it was now a pure PaaS application. New administration and configuration tools were architected so that they could be modified without having to make root level code changes. This allowed the Twitter handle to be updated on the fly, without redeploying the application. In fact, this new AzStudio-boosted version is ripe for extending functionality. With only another 1-2 hours of work incorporating existing AzStudio functionality, you'd extend the capabilities of the application far beyond the original application (e.g. to monitor multiple Twitter streams, configure alerts, deploy reports and scale much more readily).

The application had been significantly improved with little effort and time on the coder's part. AzStudio-driven improvements included:

- Better logging, error-handling, and transient fault-handling due to the AzStudio framework
- PaaS resources were more secure and configured correctly
- Code was both testable and self-healing (if code called a resource that didn't exist, AzStudio's proprietary 'Configuration-as-a-Service' [CaaS] code would create it instantaneously)
- Queue overflow and scalability were now configured in multiple dimensions (volume of traffic in queue, federation, size of items in queue, etc.)
- Reporting tools, graphs, etc.

Not only that, this new application was, overall, more secure than either the original or the manually-written version because AzStudio meticulously and clearly walks the developer through security best practices and configuration protocols. All data is encrypted in transit and much of it is encrypted at rest; therefore, when the developer needs to change passwords, configurations, etc., they are changed directly from AzStudio. This effectively locks down crucial data and code while limiting access to only what is needed.

## Conclusion – 65% Time Savings and a Better Quality App with AzStudio

AzStudio went toe-to-toe with raw Azure APIs and won.

And it wasn't just by speed alone—although a 65% cost/time savings is impressive by itself. In congruence with its speed-to-delivery, AzStudio allowed the developer to seamlessly (and nearly effortlessly) add a spectrum of new features and security improvements that the manual process would not have been able to accommodate. Intrinsically, AzStudio allows developers to skip the PaaS learning curve and the headaches of indiscriminate research by providing a set of common standards and minimizing time wasted on incompatible or foolhardy code.

In short, our experiment was able to quantitatively prove that the AzStudio framework was not only more efficient, it was more beneficial to the developer and end users as well.

*"AzStudio has a set of features that provide each developer on your team with a framework so that they don't have to trudge through all of the unnecessary preliminary preparatory work and research. It simply sets them on a path using Microsoft's suggested best practices for Azure from the beginning,"*

*—Ed Hunkin, COO and founding member of Monza Cloud*

## Want to Try Testing This for Yourself?

*If you're still one of the skeptics, feel free to replicate the experiment for yourself. Contact us for evaluation licenses so that you can compare the modernization techniques yourself. However, please ensure that your modernized application with AzStudio largely matches the specifications of our own. (If you continued to add the myriad of features available in AzStudio during the modernization process, you would greatly improve the app, but it would become very difficult to measure a direct time/cost comparison in the two efforts).*

MONZA CLOUD

WWW.MONZACLOUD.COM

## Looking to modernize your legacy applications without the time, cost, and hassle of a direct conversion to Azure PaaS?

AzStudio will increase your speed-to-delivery and provide you with a more robust and secure end product. Let's talk about incorporating AzStudio into your next modernization effort.