Integrating Quantum Computing in Legacy Mainframe Applications to Improve Security

Edward Tran
Azure Core Engineering Team
Microsoft

September 2022
Contact us at: legacy2azure@microsoft.com
Overview

Data is the foundation for many large enterprises. For many companies around the world, their future depends on using their data effectively. However, data breaches are becoming an ever-increasing threat. Protecting data from internal or external corruption and illegal access has become indispensable to reducing financial loss, reputational damages, brand harm, and other liabilities, especially for those organizations who are working with personal identifiable information (PII) data. Customers who have historically remained on their on-premises data centers or have data in legacy platforms, want to adopt newer technologies in the cloud and unlock the data, but security concerns remain on top of their minds. This creates a barrier for many enterprises, that are still using legacy mainframe systems, to migrate to Cloud. While legacy platforms like mainframe systems are reliable in security, they lack the advancement of technology and innovation, like the emergence of quantum computing, which has improved the way we encrypt and secure data. In this paper, we will demonstrate how to integrate quantum computing into an existing independent software vendor (ISV) capability to enhance the security of PII data when migrating legacy workloads to the Azure Cloud.

Quantum Computing

Quantum computing is the future of computing. It harnesses the laws of quantum mechanics to perform tasks and process information in a way that classical computers were not designed for, such as simulating complex quantum systems, searching through large unstructured datasets, and predicting the behavior of complex systems to find new and better solutions. Quantum computing, however, is not a replacement for traditional computing — instead, it’s an entirely new paradigm.

Unlike traditional binary-based classical computers, quantum machines can perform hundreds of gigabytes of calculations per second with virtually unlimited memory capabilities. It is a new approach to processing information using quantum bits, also known as qubits. Qubits have special quantum physics properties that give them the ability to solve complex problems much faster than classical bits: superposition, entanglement, and quantum interference.

Superposition is the simultaneous existence of all possible states for a quantum particle before it is observed and measured. Quantum entanglement is the phenomenon where two or more particles are unable to be described independently. Instead, their measurement results are correlated with each other, forming a single system that influences one another. This occurs when a pair of particles begin to move in parallel and remain entangled even if they are separated by great distances. Quantum interference is the idea that quantum particles, in superposition, can be influenced by an individual particle crossing its trajectory and interfering with the direction of its path. Therefore, quantum computers have been designed to reduce interference to retain the most accurate results.
These characteristics make quantum computing very powerful - so powerful that it promises solutions for many complex challenges currently being addressed by conventional computers, like encryption of PII data.

**Azure Quantum**

Azure quantum provides an open, flexible path on the cloud to quantum computing. It combines the most advanced technologies available to offer a diverse set of services that enhances productivity, speeds up progress, and safeguards technological investments. It serves as an open ecosystem to code in quantum programming languages, such as Qiskit, Cirq, and Q#, and runs these algorithms on multiple quantum systems.

The Microsoft Quantum Development Kit (QDK) is an open-source development kit that offers a set of tools to assist in the quantum software development process. With QDK the customer can write quantum programs and run them on either real quantum hardware or cloud-based simulators using Azure Quantum services. They can also opt to customize their local environment to develop and run quantum algorithms locally by installing QDK on their local environment. Another component of QDK is Q#, which is Microsoft’s open-source programming language for developing and running quantum algorithms. So QDK not only provides local and cloud-based quantum computing simulators, noise simulators, and a resource estimator but also quantum programming languages like Q#, and ready-to-use libraries to help in keeping code at a high level. These features coupled with QDK’s extensions to Microsoft Visual Studio and Visual Studio Code open the potential of running Q# with other host programs, written in .NET language (e.g., C#), or a legacy mainframe program compiled in .NET.

**Legacy Mainframe Systems Migrations to Azure Cloud**

One of the most widely used legacy systems is mainframes, like IBM z and Unisys, that run legacy programming languages like COBOL. Despite their limited scale-out capabilities and maintenance difficulties, many enterprises are still using legacy mainframe systems for their core applications and data storage. However, for customers to access modern technology like quantum computing, these customers need to migrate to Azure Cloud. There are many methods to migrate to the cloud: rehost, re-platform, reengineer, refactor, retire, and retain. However, this paper will focus specifically on rehosting.

These customers want to retain their legacy code in programming languages, like COBOL, because they do not want to rewrite their code. For example, financial institutions, healthcare organizations, retail, and government agencies that still operate on mainframes and want to safeguard their investment in their legacy code. Rehosting allows these customers to maintain their current COBOL applications while taking advantage of new technology, like quantum computing. The process of rehosting involves moving mission-critical and core applications off the mainframe platform and migrating them to the cloud without changing the underlying program language and structure. Once the legacy programs have migrated to the cloud, they can then be recompiled and accessed via IDEs,
Integrating Quantum Computing in Legacy Mainframe Systems to Improve Security

like Microsoft Visual Studio or Visual Studio Code, just like any other .NET languages like C#. These unique characteristics create the potential for integrating quantum into legacy workloads through using QDK, which supports .NET.

**Integrating Quantum Computing into Existing Mainframe Programs Customer Workflow**

Here is the customer workflow of how these legacy mainframe applications could integrate with quantum computing:

1. **Code Migration**
   - Starting with the source code in COBOL on the mainframe, the customer moves the COBOL code from the mainframe to the cloud in a text file. The data and the code are then migrated so that it is call-compatible with ISV software, like Raincode, which enables it to be accessed and used in IDEs, like Microsoft Visual Studio or Visual Studio Code, for the same (or an upgraded) service.

2. **Project Creation**
   - Next, the customer creates a project in the IDE on the virtual machine that is hosted on Azure. With Raincode’s extension to IDEs, the customer can then import the code into a Visual Studio-compatible file, which allows the customer to compile and debug the code in .NET using the COBOL compiler. Ultimately, this allows the legacy COBOL program to integrate natively with other .NET languages.

3. **Quantum Integration**
   - To gain access to quantum, the customer needs to install the Quantum Development Kit (QDK) onto their IDE. QDK introduces quantum programming languages, like Q#, Qiskit, and Cirq, and allows customers to integrate quantum algorithms with .NET. Once a program written in a quantum programming language is completed, the customer can run the quantum code in a quantum simulator or quantum hardware. Quantum simulators are classical computing that employs quantum algorithms on non-quantum computers that simulate the behavior of native quantum systems. This allows the customer to run and test their quantum program before using actual quantum hardware. However, once they are ready to run their code on quantum hardware, they can use Azure Quantum to access quantum computers remotely.
Raincode

There are several ways customers can rehost their mainframe system and recompile their COBOL program. One of those ways is using Raincode, an ISV, which provides customers with a rehosting solution for legacy systems to Azure using .NET without interrupting their ongoing operations. Specifically, Raincode has a COBOL Compiler, equipped with Microsoft Visual Studio and Visual Studio Code, that allows legacy programs to run mainframe services natively in .NET. This allows COBOL programmers to continue operating on already existing COBOL programs while newer developers can integrate other .NET languages to access new functionalities and technology.

An existing demo environment that Raincode provides is called Raincode 360, which is an advanced showcase application with sample code for customers to test mainframe applications on Azure. One of these demos on Raincode 360 demonstrates how a COBOL program can call a C# routine in a transparent way and how the C# routine can use a helper class to make sense of the data being passed by the COBOL program. This paper will demonstrate how we can leverage this existing Raincode 360 demo to integrate quantum computing in legacy mainframe applications running in COBOL.

Quantum-Mainframe Integration Design

Here is the design schematic for integrating quantum in COBOL programming:
Once the legacy COBOL modules on the mainframe systems have been recompiled with the Raincode COBOL compiler, the COBOL program can then communicate with other .NET languages as if it were COBOL.

1. **Main.cs**: the main file creates a dictionary function, provided by Raincode 360, mapping the legacy COBOL data types to the C# data types, and allowing the COBOL program to call the C# program.
2. **Legacy.cob**: the legacy file calls the C# program and passes in its data structure.
3. **Csharp.cs**: the C# file takes in the data structure and processes it, extracting the correct parameters and converting the data structure to the appropriate type. It then calls the quantum simulator to run the quantum program in the Q# file passing in the processed data structures.
4. **Qsharp.qs**: the Q# file runs a quantum operation, like an encryption algorithm, written in a quantum programming language like Q#

**Quantum Cryptography**

Once quantum computing is integrated into legacy applications, it can be leveraged to improve security and cryptography through quantum key distribution (QKD) and quantum machine learning. QKD is a quantum secure communication method that leverages quantum mechanics to enable two parties to produce a shared random secret key known only to them, which can be used to encrypt and decrypt messages. QKD specifically uses quantum superposition or quantum entanglement to transmit information in quantum states. This communication system can detect eavesdropping so that a key is produced when it is guaranteed to be secure, which is when an eavesdropper has no information about the key.

The second quantum method is quantum machine learning. Training deep models grows exponentially with the complexity and volume of data, which is why it’s important to have the computing power to develop rigorous algorithms for situations like cybersecurity protection. Quantum machine learning can enable exponentially faster and more efficient computing to develop effective algorithms for analyzing threats, responding quickly to security issues, leveraging automated threat detection, identifying potential data breaches, and defeating novel cyberattacks. These are only two specific ways that quantum computing can be used towards quantum cryptography, and there are many more.

**Future Work**

The quantum-mainframe integration described above relies heavily on utilizing Raincode, specifically their COBOL compiler that integrates .NET languages, like C#, with legacy programs. Because this implementation leverages Raincode to reconfigure the COBOL data to map with .NET data types, it utilizes a .NET language, C#, as an intermediate program to call the quantum algorithm.
In the future, quantum algorithms and programs can potentially be integrated directly into legacy programs, which may require assemblies in C# that are integrated into the Q# program. The benefit of this integration is that it bypasses the need for a separate C# program file and requires fewer intermediate program files; it would only need a C# assembly library that would be included with the Q# module. Here is an example design of how this might look:

Other Potential Use Cases

While this paper focuses on the application of quantum-legacy integration to improve security, there are many other use cases of quantum integration with legacy code. These use cases include accelerating machine learning and data analysis tasks, searching big data and locating connections faster, and enabling proactive fraud risk management. Additionally, for many financial institutions, quantum computing can be applied to many financial models and even algorithmic trading. These potential use cases motivate business executives to decide to come off the legacy mainframe systems and host their business-critical applications in more modern secure platforms like Azure Cloud.

Conclusion

The integration of legacy mainframe applications and quantum computing on Azure Cloud addresses one of the main barriers to adopting Azure Cloud—security concerns. Integrating quantum computing in legacy mainframe applications using the capabilities of Raincode allows the customer to retain their investment in COBOL programming while opening the possibility of adopting advanced technology to improve speed and security. This gives the customer more flexibility and a stronger cybersecurity system to safeguard PII data and meet regulatory guidelines like General Data
Protection regulations. Additionally, by migrating to Azure cloud, the customer achieves IT cost savings because the total cost of ownership (TCO) of the cloud’s subscription-based, the usage-driven cost model is far less expensive than mainframe systems.

Learn More

Learn more about Quantum Computing and Azure Quantum:
- Introduction to Quantum Computing
- Azure Quantum Documentation
- Q# Quantum Programming Language
- Quantum Development Kit

Learn more about Raincode and their tools:
- Raincode User Guide
- Raincode 360
- Raincode COBOL Compiler

Learn more about Mainframe Migrations:
- Mainframe and Midrange Migration
- Mainframe Migration Overview