SOLUTION BRIEF

AI/ML Modernization for IBM Mainframe Assembly

IBM Z/OS ASSEMBLER TO CLOUD-NATIVE MICROSERVICES

The task of Digital Transformation can be daunting. There are many points of difficulty and potential failures, but the outcome of avoiding modernization is far more damning. At Ionate, we are transforming modernization into a fully achievable path with Ionate AppDate.

We modernize legacy systems using AI / ML to aid in acceleration and guarantee a higher degree of success and custom results in automation for your business rules. Enabling us to produce the correct end-game product with cloud-native microservices faster, safer, and cost effectively.

Our AI / ML library is constantly growing and learning, increasing our comprehension of all legacy systems and how best to approach each one. This solution brief is designed to show how we approach a specific legacy system, IBM z/OS Assembler to Cloud-Native Microservices.

LIMITATIONS OF IBM Z/OS ASSEMBLER

Business programs started using assembly when a lot of services that we take for granted were not present.

- No databases for storing data (persistance)
- No high-performance IO to perform efficient file manipulation
- No ability of user programs to handle huge file sizes
- No ability of user programs to sort / access records in huge files
- No efficient way of sharing working data between different programs

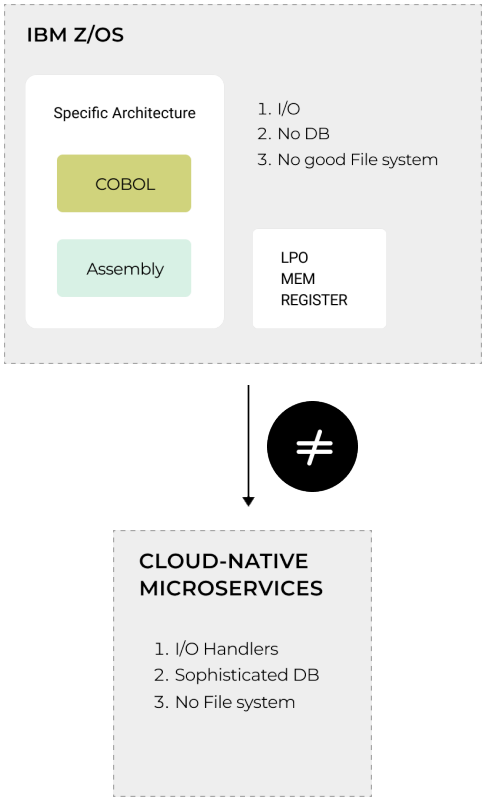
Assembly, coupled with low-level operating system services (VSAM, DFSMS) was the defacto way of solving these problems. The typical methodology was:

- Implement the business logic in COBOL
- Implement some of the performance critical sections of the business logic in Assembler
- Either Compile/Link the COBOL and Assembly programs so as to support direct calling semantics between them.

OR

- Use CICS to Orchestrate between the COBOL and Assembler programs

ASSEMBLY DISPARITY



Tenets of Modern Architectures and Features of Modern Infrastructure

`Statelessness —` As opposed to Legacy/Monolithic applications, Microservices are stateless. Being stateless not just simplifies the application design but also improves scalability.

`Having sophisticated persistance functionality —` Modern Relational and NoSQL database systems support sophisticated persistance functionality including features such as transactions, atomicity, reliability, consistency, backup/recovery, querying, which were not available in older monolithic systems.

`Stream processing infrastruture —` Modern infrastructure supports processing enormous amounts of data via streaming and there are many stream processing applications that can be used for this purpose.

`Decentralization —` Microservices are decentralized by design which improves the fault tolerance as well as minimization of SPOF (Single point of failure) risk.

`High performance IO —` Multi-threaded file systems, asynchronous IO and performance gains in storage hardware lead to very high processing IO.

`Scalablity —` By being stateless, Microservices lead themseves to being naturally more scalable.

`Caching —` Further performance gains can be achieved by applications such as Redis, which can be used not just for caching but also in edge computing.

ASSEMBLER CONVERSION PROCESS

To convert assembly, we take a four step approach where we transpile, analyze, model (AI / ML), and transform.

1. Transpile

Ionate's HLASM (High Level Assembler) transpiler works in multiple stages. In the first stage, the transpiler will parse the input assembler language, building a sophisticated internal representation of the business logic in the same. This includes the code and labels in the code segment and the symbolic variables defined in the data segment.

From this, the transpiler also generates high level code structure and data flow graph for the assembler application.

2. Analyze

The transpiler analyzes the code structure, paying special attention to usage of various system services that are used such as VSAM, Control Block and DFSMS.

Ionate's runtime includes an emulator for the System 360/370/390 HLASM language. This includes a virtual modelling of the CPU, registers and memory in Java. This allows Ionate to model any assembler instructions accurately and thus preserve the business logic.

3. Model

Ionate's goal is not to emulate every single low-level facility that is provided by zOS operating system.

Ionate instead attempts to glean and convert the high level business logic from the programs and utilize our own libraries and layers for performing system operations such as persistance, file operations and other high performance IO.

During the modelling phase, the transpiler creates a model of the intermediate representation that reflects the high level business logic and purpose of the code, as opposed to low level instructions.

4. Transform

The final phase is transformation in which the high level intermediate representation is transformed into the final Java code which can be exposed as a REST webservice or an asynchronous job as needed.

> **For more information, please reach out to sales@ionate.io or visit our website at www.ionate.io**