lustratus

Research

Mainframe API tools: Competitive Assessment

Comparing vendor solutions for API-enabling the mainframe

Author: *Steve Craggs*

February 2019
Version 1.00

Table of Contents

Executive Summary

The API Economy is booming, and with such huge historic investments in mainframes it is vital for companies to find ways to bring these assets into the API world. The Lustratus report "Best of Breed API Middleware for Mainframes" examined the challenge of API-enabling mainframes and developed a checklist of key functions for buyers to use in discussions with prospective suppliers. It identified the following API software segmentation, distinguishing between generic API Management middleware and mainframe-specific API middleware.

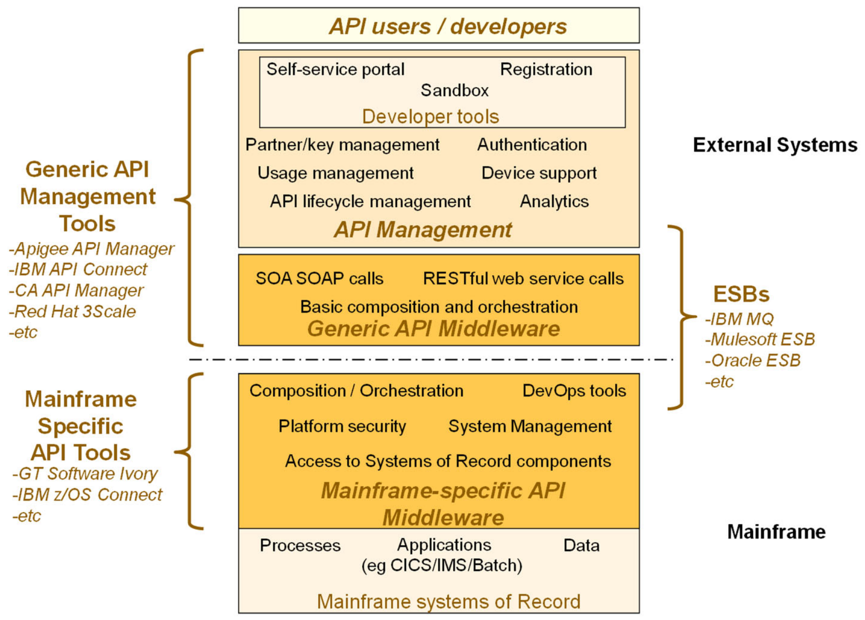
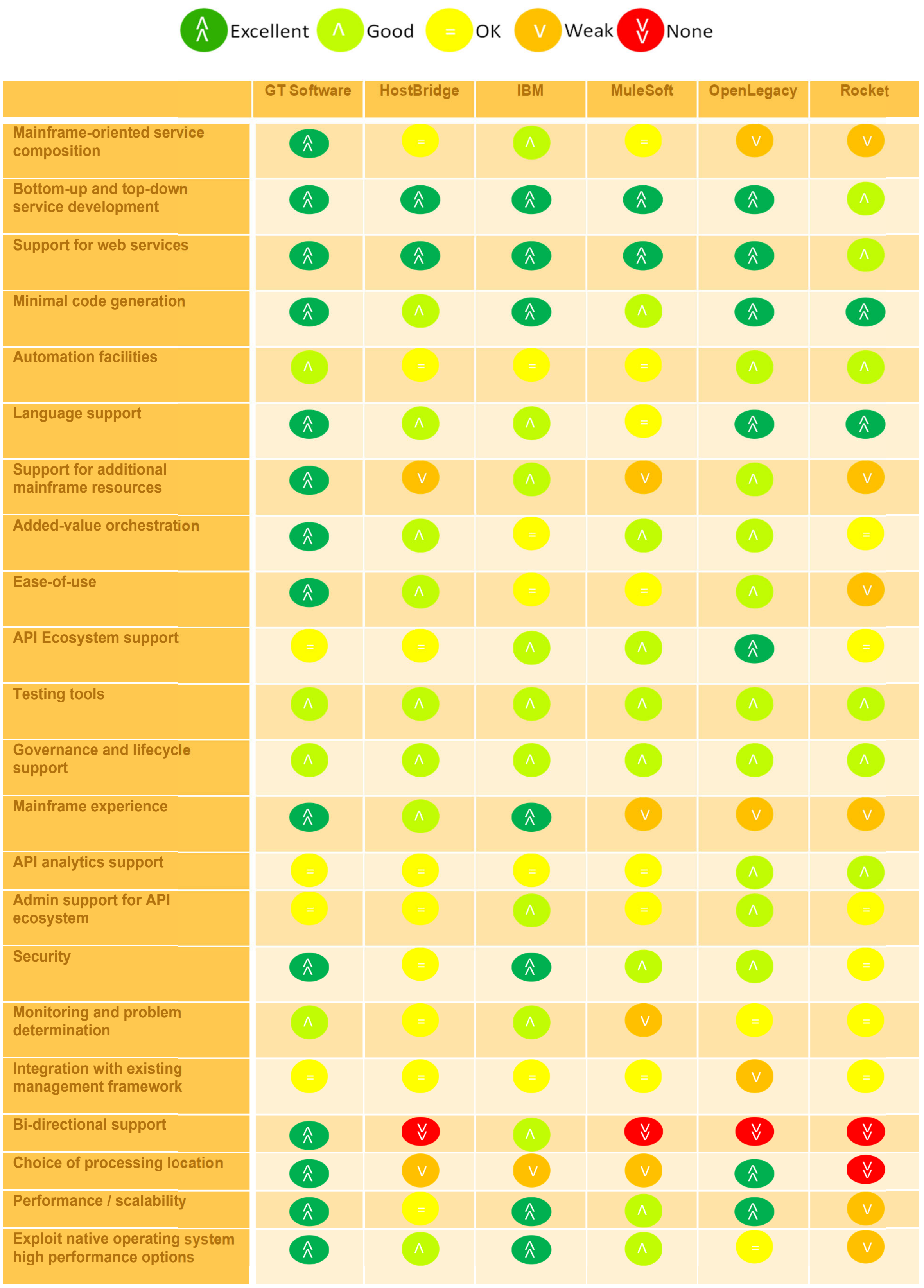


Figure 1: The Lustratus view of API-enablement software for the mainframe

This assessment looks at six different vendors of mainframe API middleware, covering what each offers and providing a view on how well each addresses the various checklist items. The focus of this assessment is primarily on the mainframe-specific API middleware rather than the more generic API management middleware. Some vendors could be classed as API Management vendors that also provide some mainframe connectivity. Others are specialist mainframe vendors that usually have partnerships with one or more API Management vendors to cover the generic requirements. There are even some that try to cover both areas.

Finding the best match will depend to a large extent on the mainframe API requirements the buyer wants to address. For example, if the target is to make available a small number of fairly simple CICS transactions with a modern look and feel on various devices, this can be achieved relatively easily with little need for a mainframe-skilled partner. However, if various mainframe environments such as CICS, IMS, IDMS or Batch have assets to be exposed as APIs, spanning simple queries to complex, conversational transactions, the mainframe skills and experience of the chosen supplier is likely to be far more important.

The assessment is highly subjective and not intended to be definitive in any way, but rather it aims to guide buyers on some of the questions to ask as they evaluate the different solutions. The table below indicates the level of support of each vendor, using the following scheme:


ΛΛ Excellent Λ Good = OK V Weak VV None

	GT Software	HostBridge	IBM	MuleSoft	OpenLegacy	Rocket
Mainframe-oriented service composition	ΛΛ	=	Λ	=	V	V
Bottom-up and top-down service development	ΛΛ	ΛΛ	ΛΛ	ΛΛ	ΛΛ	Λ
Support for web services	ΛΛ	ΛΛ	ΛΛ	ΛΛ	ΛΛ	Λ
Minimal code generation	ΛΛ	Λ	ΛΛ	Λ	ΛΛ	ΛΛ
Automation facilities	Λ	=	=	=	Λ	Λ
Language support	ΛΛ	Λ	Λ	=	ΛΛ	ΛΛ
Support for additional mainframe resources	ΛΛ	V	Λ	V	Λ	V
Added-value orchestration	ΛΛ	Λ	=	Λ	Λ	=
Ease-of-use	ΛΛ	Λ	=	=	Λ	V
API Ecosystem support	=	=	Λ	Λ	ΛΛ	=
Testing tools	Λ	Λ	Λ	Λ	Λ	Λ
Governance and lifecycle support	Λ	Λ	Λ	Λ	Λ	Λ
Mainframe experience	ΛΛ	Λ	ΛΛ	V	V	V
API analytics support	=	=	=	=	Λ	Λ
Admin support for API ecosystem	=	=	Λ	=	Λ	=
Security	ΛΛ	=	ΛΛ	Λ	Λ	=
Monitoring and problem determination	Λ	=	Λ	V	=	=
Integration with existing management framework	=	=	=	=	V	=
Bi-directional support	ΛΛ	VV	Λ	VV	VV	VV
Choice of processing location	ΛΛ	V	V	V	ΛΛ	VV
Performance / scalability	ΛΛ	=	ΛΛ	Λ	ΛΛ	V
Exploit native operating system high performance options	ΛΛ	Λ	ΛΛ	Λ	=	V

Figure 2: Mainframe API vendor comparison table

Introduction

Before looking at the vendor solutions, it is worth taking a high level look at the three main approaches generally used to solving the problems of exposing mainframe applications as APIs:

- Screen-scraping
- Enterprise Service Bus (ESB)
- Programmatic access (Connectors / Adapters)

Vendors will generally offer one or more of these approaches. Choosing the most appropriate approach will depend on the mainframe assets that need to be exposed as APIs.

Screen-scraping

Screen-scraping is one of the earliest forms of opening up mainframe applications to external applications. In simple terms, since many mainframe applications operate with 3270 screens, by driving a 3270 data stream an external program can appear to the mainframe as an actual 3270 screen. Normally, the client side will present the user with a different interface, for example for mobile devices, and then when the input data is entered this is then mapped to the desired 3270 data stream to drive the required mainframe application. A simple example might be a CICS transaction to return customer data from a customer number. A client application can use any UI it wishes to gather the customer number, then map the information into a 3270 stream, execute it on the mainframe, gather the responding 3270 stream and map it back to the local UI.

The obvious advantage of this approach is that the mainframe is totally unaware that the mainframe application is being driven externally; it thinks it is talking with a local 3270 screen. No changes are required to mainframe applications at all. However there are a number of major downsides to this approach. If the mainframe application is heavily conversational, for instance, every interaction requires a trip back to the user location and then return to the mainframe, together with the mapping of a new data stream. The result is that performance suffers dramatically, and the approach lacks in scalability. Screen-scraping is best suited to situations where there are a small number of simple in/out transactions that need to be exposed as APIs; for anything beyond that, it is likely to prove brittle and a drag on performance.

ESBs

The whole point about an Enterprise Service Bus (ESB) is that it provides efficient connectivity between different systems. As such, it can certainly be used to provide connectivity between mainframes and external systems. However, once into the mainframe, it has to have some way to drive the desired mainframe activity. One way to achieve this is to 'message-enable' the mainframe applications, so they can be driven through the receipt of ESB messages and return output the same way. This can be an efficient and scalable way to operate in execution terms, but it does require the mainframe applications to be altered to message-enable them. This can be quite a complicated task, and requires a range of mainframe specialist programming skills.

In order to make the job easier, ESB vendors typically offer a range of adapters or connectors that are designed to key off the messaging provided by the ESB but then interoperate with a target application or environment. Some of these adapters / connectors may be for mainframe systems such as CICS. Using these provides another way for accessing mainframe applications and resources, but typically will still require work to fit the mainframe applications to the provided adapter interface. Connectors and adapters are specific examples of the third category, programmatic access.

Programmatic access

The programmatic access approach to accessing mainframe applications and data is the most flexible and powerful. Most mainframe systems such as CICS and IMS have various means of driving activities programmatically. For example, CICS applications often support some form of external call or link access, usually using the COMMAREA to provide parameters to the CICS application and fields for the response. By using these programmatic interfaces, programs can be driven efficiently and effectively. An external client makes the request from its UI, the input parameters are mapped to those required for the application, the application is executed and the answer returned. Once the framework to operate in this fashion is set up, often all that is required to drive a new program is for a developer to specify how the data fields map between the client (for example JSON) and the target environment (eg a CICS COMMAREA).

This approach has some distinct advantages. Assuming the target applications are already enabled for some form of programmatic access, no further changes to the programs are required. Since this mechanism is actually the way a lot of CICS programs interact anyway, many of them will be designed to support this. This approach is flexible and generally scalable. One exception though is that for access to a string of mainframe activities, some of the same scalability/performance issues encountered with screen-scraping may arise in that there will be potentially frequent trips to and from the mainframe with the corresponding mapping from one format to the other and back again. Some vendors however will provide some sort of mainframe-resident agent that can carry out a chain of different mainframe applications before returning with the answer.

The final point to note is that providing a comprehensive programmatic access solution will almost certainly require the vendor to have extensive mainframe experience and skills.

APIs and the OpenAPI standard

The three approaches listed each provide evolving ways to access mainframe resources and applications, but so far there has been no mention of APIs. In simple terms, the API technology is what makes these often highly technical mainframe access options consumable in many different environments and with minimal mainframe skills.

This challenge has been tackled before, most notably with the concept of SOAP-based web services. With SOAP web services, the web service consumer can gain the required information to drive a mainframe-based service through the WSDL description of the service. For some mainframe operations, SOAP-based web services may still be the most desirable option, particularly where the rigorous nature of SOA message structures are required for such things as message encryption. But the less complex, simpler approach is to use RESTful APIs. The new OpenAPI standard provides the way to describe and invoke the API, making it simple to invoke the mainframe activity from different platforms and technology bases. Of course, technology will still be required under the covers based on one or more of the previously listed mainframe access approaches, but this can all be done by the API middleware transparently to the consumer of the service.

Bearing these approaches in mind, the next topic is to look at the six vendors of mainframe API-enabling tools in more detail. The vendors considered are (in alphabetical order):

- GT Software
- HostBridge
- IBM
- MuleSoft
- OpenLegacy
- Rocket Software

GT Software

GT Software has spent the last three decades working with IBM mainframe customers to help them get the best out of their assets. As a mainframe-based company it offers a range of mainframe-specific products and packages, but the key one in terms of the scope of this paper is Ivory Service Architect, more commonly referred to simply as Ivory. This product suite is designed to leverage and optimize mainframe assets within the wider IT world, turning mainframe system of record applications, processes and data into APIs and enabling these systems of record to utilize APIs themselves.

GT Software' Ivory Service Architect fits within the Lustratus API architecture as follows:

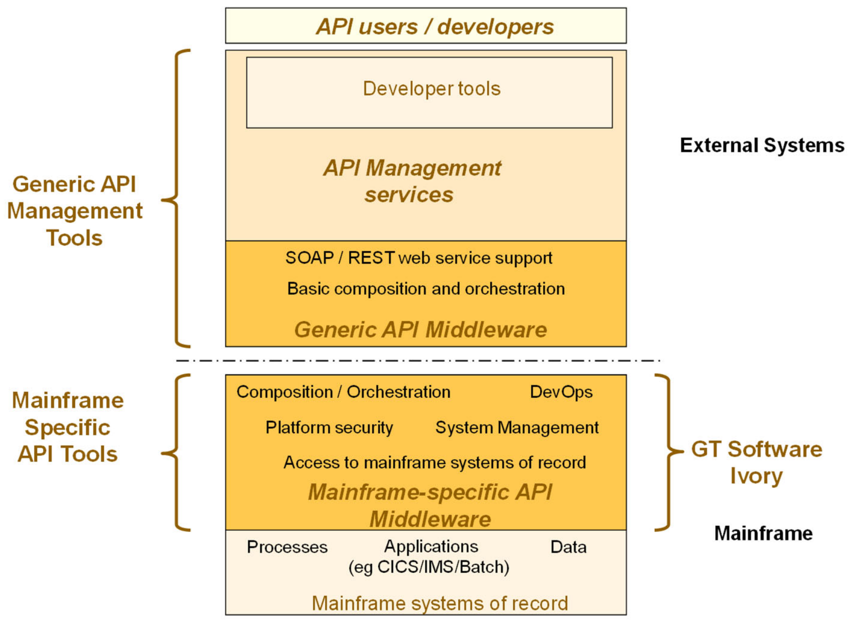

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Basic composition and orchestration
Generic API Middleware
Generic API Management Tools
External Systems
Composition / Orchestration
DevOps
Platform security
System Management
Access to mainframe systems of record
Mainframe-specific API Middleware
Mainframe Specific API Tools
GT Software Ivory
Mainframe
Processes
Applications (eg CICS/IMS/Batch)
Data
Mainframe systems of record

Figure 3: GT Software Ivory's position in the mainframe API architecture

Mainframe API functionality

Ivory is a set of tools for making existing mainframe applications available to other users, either externally or within the mainframe. Ivory services are created using a graphical IDE, Ivory Studio, to produce metadata describing the desired functionality. Once created, these Ivory services can now be deployed as REST/JSON or SOAP based services, or as services to be invoked by other mainframe applications, all under the control of the Ivory runtime component, Ivory Server. The SOAP services are documented with WSDL and the REST/JSON ones with OpenAPI.

Ivory Studio is PC-based while Ivory Server can be deployed locally on the mainframe or externally on a distributed platform. The choice of Ivory Server location will depend on considerations like mainframe load restrictions and the underlying structure of the services. For the mainframe, Ivory Server can run within CICS, in its own region or in Linux on Z (IFL). Externally, it can run in Windows/Java or Linux/Java environments or in an OpenShift or Docker container.

Ivory can build APIs for CICS, IMS, 3270, Batch, IDMS and Natural mainframe systems, and it also provides orchestration support for building composite APIs from multiple systems of record applications. This orchestration support provides the flexibility to optimize the API, for example to reduce traffic between different systems. Ivory Function support, a light scripting capability, broadens orchestration and control options. The API creation process is carried out using wizards in Ivory Studio. Copybooks describing the data inputs and outputs to the systems of record can be mapped graphically to the API formats, and a drag and drop facility provides the orchestration support. Access is achieved using one of several mainframe features such as CICS COMMAREAs / Channels and Containers, IMS Connect, or even direct 3270 access where screen-scraping is desirable. This 3270 support includes a recording wizard to turn existing 3270 flows including BMS maps into Ivory metadata. Ivory also provides testing tools for both unit and regression testing, and also supports open source tooling such as Postman, SoapUI and SwaggerUI.

Security depends on whether the Ivory service is deployed as a SOAP or RESTful web service. In the SOAP case Ivory uses the security information from the SOAP header, while in the REST/JSON deployment Ivory supports the use of JSON Web Tokens (JWTs). The JWT is generated on authentication and then passed for subsequent service requests.

	Comments
Mainframe-oriented service creation	Comprehensive support eg CICS Channels and COMMAREAs, IMS conversational and non-conversational, 3270 access, batch
Bottom-up and top-down service development	Unusually, Ivory provides equally effective support for both top-down and bottom-up design methodologies
Support for web services	Ivory supports REST-based and SOAP based web services, with interfaces described in OpenAPI and XMLdocs
Minimal code generation	Almost everything in Ivory is done without any code generation. The only code actually produced is XML for internal Ivory use
Automation facilities	Automation support includes a light scripting facility as well as input/output exits for validation etc.
Language support	Support for PL/1, COBOL
Mainframe systems coverage	Comprehensive coverage, including CICS, IMS, 3270, batch, IDMS, data sources
Added-value orchestration	Ivory offers a fairly standard graphical orchestration / workflow specification tool for Windows, all as part of the one product
Ease-of-use	The wizard-based approach and the lack of code generation combine to make Ivory particularly quick and simple to use
API Ecosystem support	Basic support
Testing tools	Test facility for Ivory services and orchestrations, including support for a wide range of open source test tools like Postman, SwaggerUI and soapUI
Governance and lifecycle support	Support for bringing Ivory services online or offline
Mainframe experience	As 30-year veteran mainframe specialists, GT Software offers a detailed

	understanding of mainframe technology with all its intricacies and quirks
API analytics support	As provided through the underlying mainframe systems eg CICS, IMS, MVS
Admin support for API ecosystem	Start and stop of Ivory services
Security	Support for SOAP-based security and also JSON Web Tokens; also uses security features in the underlying mainframe systems
Monitoring and problem determination	Logging is through the underlying mainframe systems. The Ivory test facility provides an environment for testing and tracing Ivory service flows and orchestrations
Integration with existing management framework	Through underlying mainframe systems
Bi-directional support	Strong bi-directional support including external and internal consumption of services
Choice of processing location	Comprehensive support for running Ivory execution in CICS, batch, other z/OS partition or off-mainframe
Performance / scalability	Optimized performance through choice of processing location, such as in CICS, batch, zIIP offload
Exploit native operating system high performance options	Ivory can use port sharing in a multi-server environment to provide high-performance throughput and failover support

Figure 4: GT Software Ivory Service Architect assessment

HostBridge

HostBridge was founded in 2000 with the main aim of helping IBM mainframe CICS users to leverage their CICS programs and transactions in non-mainframe environments and distributed environments. The HostBridge offerings are examples of mainframe-specific integration middleware. In mainframe API terms, the focus area for this assessment, this means that HostBridge aims to make CICS transactions available as web services that can then be used by generic Application Managers to create APIs.

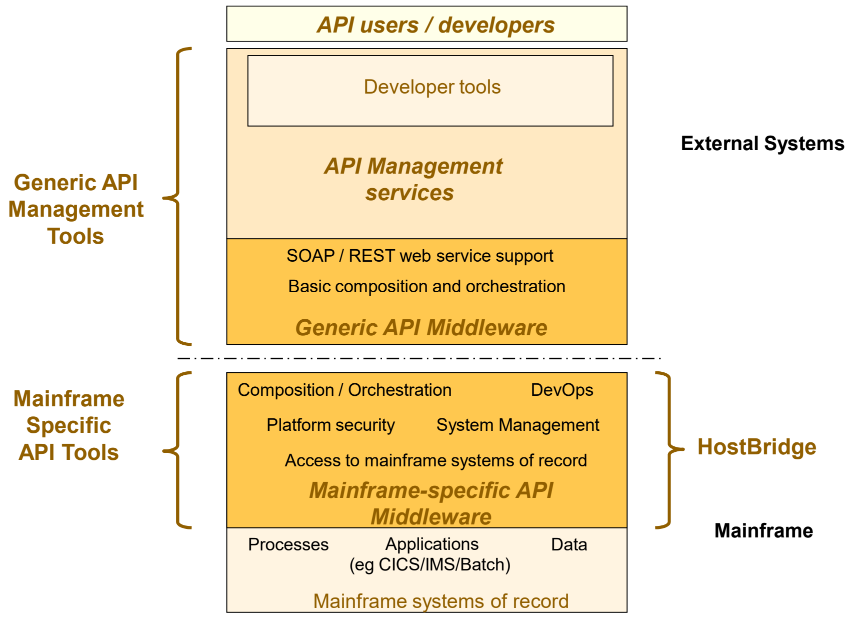

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Basic composition and orchestration
Generic API Middleware
Generic API Management Tools
External Systems
Composition / Orchestration
DevOps
Platform security
System Management
Access to mainframe systems of record
Mainframe-specific API Middleware
Mainframe Specific API Tools
HostBridge
Mainframe
Processes
Applications (eg CICS/IMS/Batch)
Data
Mainframe systems of record

Figure 5: HostBridge's position in the mainframe API architecture

HostBridge products

The main HostBridge product is HB.js. This is the main engine for developing, composing, deploying and operating CICS transactions as web services or through Javascript scripts. HostBridge also offers HB Base XML which converts CICS transactions to XML, so that calling applications can easily access CICS data in the flows.

HB.js includes a runtime that runs inside a CICS region on the mainframe together with a development environment, HostBridge Eclipse IDE. Using the IDE, the user can use a screen-scraping approach for terminal-oriented CICS transactions or a more programmable method for COMMAREA-based access or for DB2, VSAM or DL/I data. The screen-scraping approach is a bit more advanced than typical screen scrapers of the past in that it uses metadata to decouple the field contents from the positions on the screen based on the identifier for that particular screen. This means that a simple change to field positioning on the screen does not require a change to the linkage. In addition, the ability to orchestrate flows within the mainframe means that conversation-heavy CICS transactions can be handled without constant to and fro trips from the mainframe to an external mapper and back again as is the case for more primitive screen scraping offerings. HB.js is not limited to the screen-scraping approach though. It also supports driving CICS programs through COMMAREAS and can interoperate with IBM MQ.

If the user is interested in screen scraping, HostBridge offers a Transaction Explorer tool in the IDE which replicates the CICS screen flows from a live CICS transaction, assisting with navigating and planning the screen-scraping solution. HostBridge IDE also includes a test service which can be run in conjunction with Postman as a way to test and debug web services, although this of course requires separate product downloads. As far as security is concerned, running in CICS ensures that the HB.js runtime can fit in with security managers like RACF, ACF/2 and TopSecret, and it also supports SSL and digital certificates.

The process of defining and building CICS-based web services is quite complicated. It is largely done by defining tables, XML files and maps, and requires some skill to use effectively. It is also important to recognize that the HB.js runtime runs within the CIC region on the mainframe, and is designed for CICS programs and transactions. There is no option to run it off-mainframe and it is not designed to handle other mainframe environments like IMS or batch.

	Comments
Mainframe-oriented service composition	Support is limited to CICS applications
Bottom-up and top-down service development	Both supported at a basic level
Support for web services	Support for RESTful and SOAP-based web services, using JAVA class libraries in Javascript
Minimal code generation	Automatic code generation generates little code, although Javascript code is required for more complex services
Automation facilities	Since HB runs in CICS, it can benefit from some of the CICS automation facilities
Language support	All CICS languages supported
Support for additional mainframe resources	Since HB runs in CICS, all mainframe support is CICS-centric
Added-value orchestration	Composite flows can be built to bundle up mainframe processing and reduce traffic, but automation requires Javascript
Ease-of-use	Building the required maps and files requires quite a lot of manual effort and a reasonable amount of technical skill, while Javascript skills may be needed for complex services
API Ecosystem support	Running in CICS provides security and monitoring, but there is nothing outside the mainframe
Testing tools	The HB Service Test together with Postman offers a very usable and effective test harness for checking out the web services
Governance and lifecycle support	Life cycle management is supported
Mainframe experience	HostBridge is dedicated to the mainframe, so has definite mainframe skills. Having said that, these skills are primarily with CICS since the product runs under CICS
API analytics support	Nothing specific other than offered for CICS in general
Admin support for API ecosystem	Standard administration tools provided

Security	SSL is supported, and within the CICS region the runtime can work with most mainframe security managers
Monitoring and problem determination	Since the runtime runs in the CICS region, CICS tools can provide some level of monitoring
Integration with existing management framework	Through underpinning CICS region
Bi-directional support	None
Choice of processing location	The HostBridge product runtime runs within the CICS region.
Performance / scalability	Since it runs within CICS, HB.js can package up activities to avoid unnecessary traffic. The screen-scraping approach generally does not scale well though
Exploit native operating system high performance options	Support through underlying CICS functionality

Figure 6: HostBridge HB.js assessment

IBM

IBM provides its own generic API management capabilities through its IBM API Connect offering, delivering the expected set of functions for administering, securing, managing, testing, monitoring and analyzing APIs and their usage. As far as the scope of this assessment is concerned, however, the key offering is IBM z/OS Connect Enterprise Edition (EE) and its related products. The IBM z/OS Connect EE offering provides IBM's mainframe-specific API enablement facilities that produce make available mainframe systems of record APIs to its IBM API Connect tools.

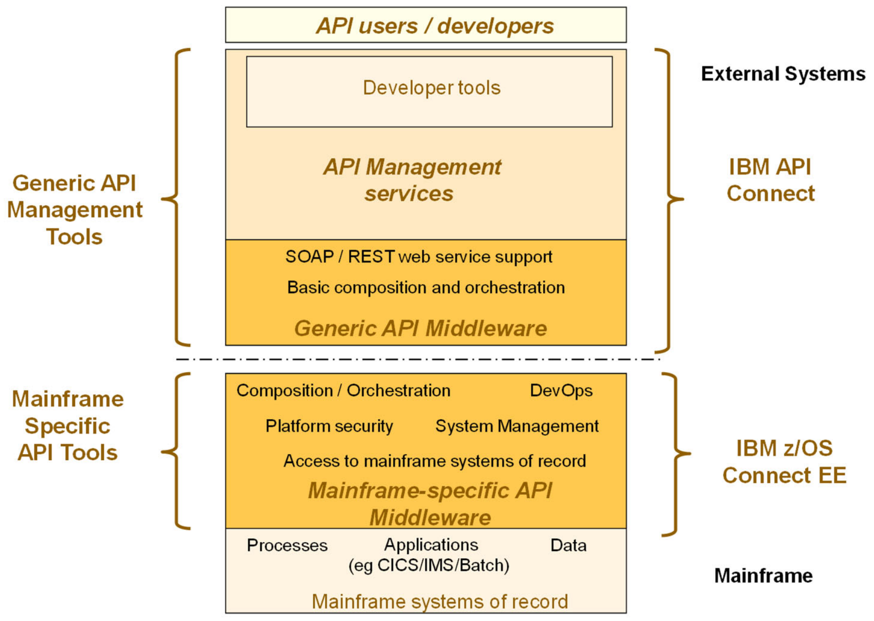

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Basic composition and orchestration
Generic API Middleware
Composition / Orchestration
DevOps
Platform security
System Management
Access to mainframe systems of record
Mainframe-specific API Middleware
Processes
Applications (eg CICS/IMS/Batch)
Data
Mainframe systems of record
Generic API Management Tools
Mainframe Specific API Tools
External Systems
IBM API Connect
IBM z/OS Connect EE
Mainframe

Figure 7: IBM API Connect and z/OS Connect positioned in the mainframe API architecture

IBM z/OS Connect EE

The IBM z/OS Connect Enterprise Edition (EE) offering is designed for building APIs for system of record applications. z/OS Connect EE runs in its own z/OS Liberty address space, and is built around the concept of 'service providers' that provide access to the SoR applications. It includes an IMS and a WOLA (WebSphere Optimized Local Adapter) service provider, as well as an SDK for building a custom service provider that can work with other mainframe environments such as third party systems. The WOLA service provider supports access to CICS and Batch. However, it should be noted that the WOLA support requires Liberty, making it another moving part to have to handle.

z/OS Connect EE requires a set of Service Archive (SAR) files for the APIs to be offered. These SAR files provide all the information needed to deploy the services and enable them as JSON assets. The z/OS Connect API Toolkit is the main development environment for z/OS Connect EE activities such as creating SAR files and defining and deploying the RESTful APIs. For CICS applications, for example, the COBOL copybook describing the COMMAREA can be imported and then redacted as required, ensuring only the required information is shown. A menu-based facility is used to create the mappings between JSON and SAR data formats. When creating the API, versioning is supported to allow existing APIs to be updated. When the API is produced, Swagger documentation is also created and this can now be used via the Swagger UI to test the API and check the resultant data flows. Once validated, the API can now be deployed directly from the API Toolkit.

At execution time, there are a number of user exits and options for added value. For example, the user can develop Interceptors using the toolkit which provide pre-processing exits. The z/OS Connect EE package includes pre-built ones for audit, authorization (LDAP, SAF) and logging of the input request. In addition, there is an exit before the JSON is transformed to the format for the system of record applications. This can be used to manipulate information as HTTP headers, for instance, to set up JSON defaults, or any other pre-transformation activities. All major mainframe languages are supported, and z/OS Connect EE also offers bi-directional support where a mainframe system of record application can call an external API using the API Requester support.

Once the RESTful APIs have been created, they can be utilized from IBM API Manager which provides the standard generic API management features like orchestration, management, monitoring, analyzing, securing and monetizing APIs. However, since this is a separate product package it increases the complexity of the solution. As a result, where API Manager is required in the following assessment, the solution has been marked down since this review focuses on the mainframe API middleware.

	Comments
Mainframe-oriented service composition	Support for IMS, CICS, DB2 and Batch. Other mainframe systems can be custom-built using the provided SDK
Bottom-up and top-down service development	Both supported
Support for web services	IBM z/OS Connect EE creates RESTful APIs, but can consume other web services / APIs
Minimal code generation	Minimal code generation
Automation facilities	There are some utilities for optimizing the service definitions and bindings
Language support	Built in support for COBOL, PL/1 and C
Support for additional mainframe resources	Broker (MQ) access also supported
Added-value orchestration	Orchestration requires the API Manager product
Ease-of-use	Various utilities carry out a lot of the manual work, but the process is still rather manual and menu-driven
API Ecosystem support	IBM Z/OS Connect EE works closely with IBM API Manager, supporting all of the main aspects of an API Ecosystem
Testing tools	APIs created by z/OS Connect can be tested directly using the Swagger UI
Governance and lifecycle support	Combined with IBM API Manager, IBM z/OS Connect EE provides strong governance and lifecycle support
Mainframe experience	Naturally, IBM has enormous mainframe experience

API analytics support	IBM API Manager provides all the analytics support for mainframe APIs
Admin support for API ecosystem	IBM API Manager and z/OS Connect EE provide full administration support
Security	SAF and LDAP security are directly supported. In addition, IBM API Manager provides more security services
Monitoring and problem determination	Mostly handled by IBM API Manager rather than IBM z/OS Connect EE
Integration with existing management framework	Through IBM API Manager
Bi-directional support	Support for calling RESTful APIs from z/OS systems is provided
Choice of processing location	IBM z/OS Connect EE runs in its own Liberty address space. The API Toolkit can run in a Windows environment
Performance / scalability	Good performance and scalability
Exploit native operating system high performance options	IBM can use any of its native options since z/OS Connect runs in its own mainframe region

Figure 8: IBM z/OS Connect EE assessment

MuleSoft

MuleSoft was founded in 2006 as an IT integration vendor. It is probably best known for its Mule Enterprise Service Bus (ESB), but it has broadened its range of offerings considerably over the last ten years. The MuleSoft Anypoint Platform is its ESB-based integration suite used for API development, and fits in the mainframe API reference architecture as depicted in the diagram below.

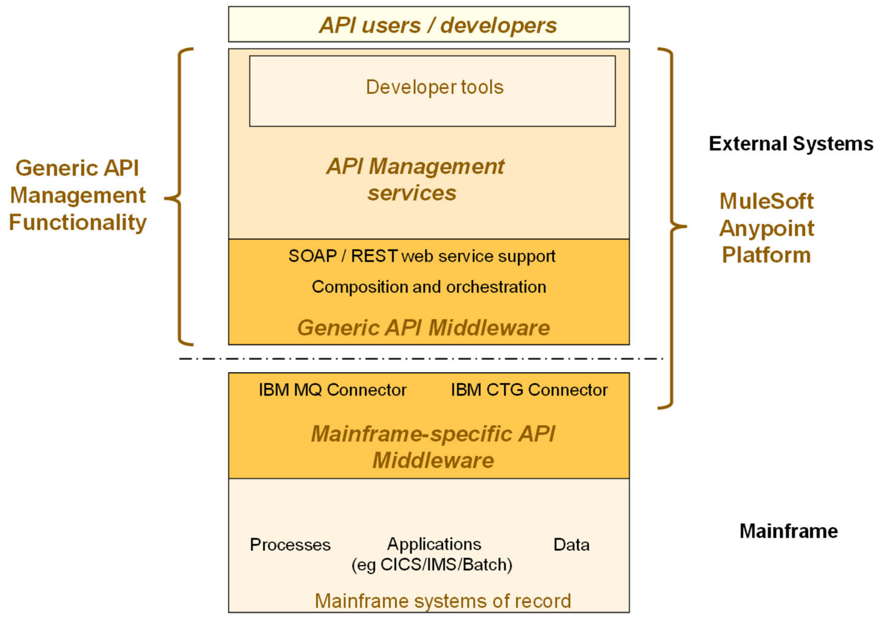

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Composition and orchestration
Generic API Middleware
Generic API Management Functionality
External Systems
MuleSoft Anypoint Platform
IBM MQ Connector
IBM CTG Connector
Mainframe-specific API Middleware
Processes
Applications (eg CICS/IMS/Batch)
Data
Mainframe systems of record
Mainframe

Figure 9: MuleSoft Anypoint's position in the mainframe API architecture

MuleSoft Anypoint

MuleSoft Anypoint is a fairly comprehensive API Management platform. It provides a wide range of generic API management tools for API development, simulation, testing, tracking and reporting. APIs can be orchestrated together using the MuleSoft Flow Designer, and Anypoint also offers analytics capabilities. It has its own DataWeave language to deal with mapping data structures between systems, and for transactional support it now uses the open-source Bitronix transaction manager in place of its original choice of the JBoss one. APIs are documented with the RAML modeling language, which is MuleSoft's preferred approach, but there is an import/export function to support Swagger.

However in terms of Mainframe API enablement, which is the focus of this research, MuleSoft provides the CICS Transaction Gateway (CTG) Connector and the IBM MQ Connector. Anypoint Connectors provide the local agent to handle interfacing with specific target environments, and the CTG Connector was provided earlier this year as an agent for integrating CICS applications, either through channels and containers or COMMAREAs. The IBM MQ Connector is an agent that can interface with IBM MQ, enabling the calling application to read or write messages to the IBM MQ queues.

The first requirement to use the Anypoint CTG Connector is to install the IBM CICS Transaction Gateway. Once this is done, the developer can configure the Connector and import data structures for DataWeave to process to transform the field information between the different formats. The Anypoint Studio tools to achieve this are fairly manual in operations. Authentication happens within the Connector using the IPIC protocol to link with the CICS regions over TCP/IP. SSL is also supported. The user can then specify whether to use CICS channels and containers or CICS COMMAREAs for the integrations, although data transfer in the COMMAREA case is limited to 32KB. If transactional integrity is required, Anypoint relies on the use of a Bitronix Transaction Manager to provide XA facilities.

The MQ Connector is rather more basic. By providing a Connector to access an IBM MQ broker, MuleSoft has enabled an Anypoint application to interface with IBM MQ at the pub/sub or listen/reply levels. This makes it possible to utilize any target application that has been IBM MQ-enabled,

	Comments
Mainframe-oriented service composition	Support is limited to CICS or MQ-enabled applications
Bottom-up and top-down service development	Both supported at a basic level
Support for web services	Support for RESTful and SOAP-based web services, with RAML or Swagger documentation
Minimal code generation	There is quite a bit of manual effort required to bring in the schemas and map them as required for using the CTG. Using the MQ Connector will require target applications to be MQ-enabled
Automation facilities	Some runtime automation support
Language support	All CICS languages supported
Support for additional mainframe resources	Support for CICS applications and any MQ-enabled resources
Added-value orchestration	Anypoint provides good orchestration support
Ease-of-use	Using the CTG is fairly manual, and for non-CICS applications they would need to be message-enabled
API Ecosystem support	Good support for security, testing, monitoring and general administration at the off-mainframe level, but nothing inside the mainframe
Testing tools	A range of testing options included, as well as simulation support
Governance and lifecycle support	Anypoint provides fairly comprehensive life cycle management tools
Mainframe experience	Until 2018, the only mainframe support was through the MQ connector. MuleSoft has little mainframe experience, although they do work closely with IBM when required
API analytics support	Anypoint has a range of monitoring and analytics offerings, although of course these are all for the off-mainframe environment
Admin support for API ecosystem	Standard administration tools provided
Security	The Anypoint CTG connector uses the IP interconnectivity (IPIC)

	protocol and also supports SSL
Monitoring and problem determination	Anypoint provides PD tools, but not specifically for the mainframe
Integration with existing management framework	Through underlying mainframe systems
Bi-directional support	None
Choice of processing location	The Anypoint CTG Connector runs on the mainframe
Performance / scalability	Anypoint provides various monitoring tools. Whether using the IBM CTG or IBM MQ Connector, scalability should be unaffected
Exploit native operating system high performance options	Support through underlying mainframe systems, eg CICS, MQ

Figure 10: MuleSoft Anypoint assessment

OpenLegacy

OpenLegacy is a very young company, founded in 2013. It offers open source software for API integration and management. OpenLegacy offers both generic application management capabilities as well as specific support for legacy applications, primarily for IBM AS/400 but also with some IBM mainframe support. The diagram below illustrates where OpenLegacy fits within the Lustratus mainframe API architecture.

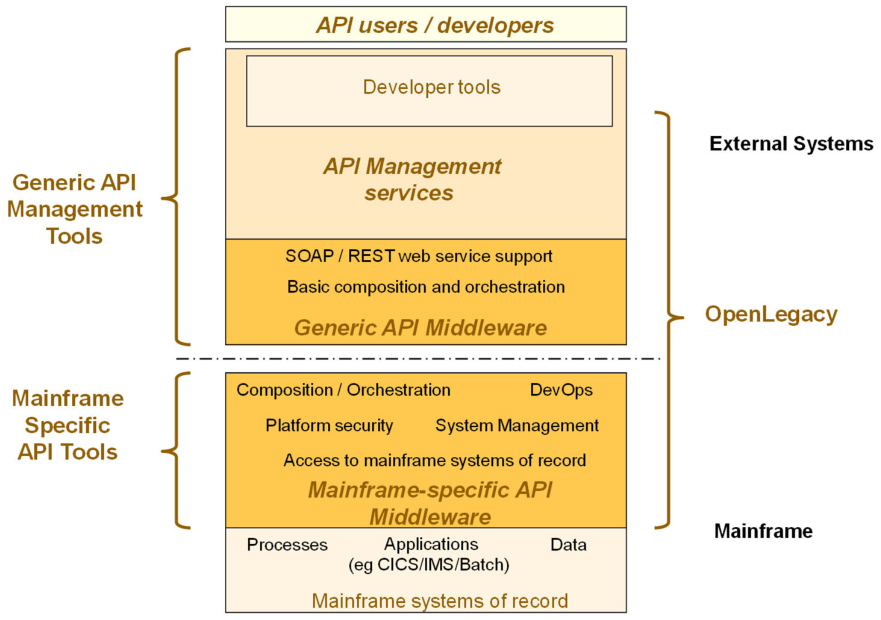

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Basic composition and orchestration
Generic API Middleware
Composition / Orchestration
DevOps
Platform security
System Management
Access to mainframe systems of record
Mainframe-specific API Middleware
Processes
Applications (eg CICS/IMS/Batch)
Data
Mainframe systems of record
Generic API Management Tools
Mainframe Specific API Tools
External Systems
OpenLegacy
Mainframe

Figure 11: OpenLegacy's position in the mainframe API architecture

OpenLegacy products

The main OpenLegacy offering is its API Software for API Integration and Management suite. As an open source project, the OpenLegacy software is unsurprisingly heavily geared to standard environments. It is based on a Java stack, embracing such initiatives as Eclipse, Tomcat, Maven, Spring, OAuth and Swagger. Its aim is to enable the user to build APIs that can be used as SOAP or RESTful web services or in Java scripts. On the API Management side it offers a management console as well as monitoring to track metrics such as API usage, and analytics to provide a more detailed understanding. The wizard-based approach used by the API Development tool leads the developer through the necessary steps to create and deploy the API.

As far as the scope of this assessment is concerned, being focused on mainframe API enablement, the key components are the API Connectors. The principle behind these connectors is to provide a way to extract key metadata from the mainframe to create a Java API. Once the metadata is gathered, the OpenLegacy software even makes it possible for the user to change the business logic in the API rather than having to return to the mainframe. OpenLegacy is also a proponent of microservice architectures, and it enables the user to package mainframe activities into the microservices. The main artefacts created by the OpenLegacy tool are POJOs (Plain Old Java Object) that can then be used within scripts or through SOAP or RESTful APIs.

When creating the APIs, the user has to specify what the source is for the core code and what form will be used for delivery. So for example the delivery destination might be a SOAP service, a RESTful service or a Java API. Mainframe application coverage includes 3270 screen-based access, IMS applications through IBM IMS Connect, SOAP or REST services and mainframe remote procedure calls (RPCs). In the CICS case for instance, this means an API could be built to access a CICS transaction through screen-scraping, a web service if one has already been created or through the CICS RPC support. OpenLegacy-created services from multiple back ends can be orchestrated together to provide composite services with the Eclipse IDE.

In general terms, OpenLegacy mainframe API enablement provides reasonable levels of automation but does have some more manual and technical areas of usage. While it has wizards to carry out tasks like extracting COMMAREA information from COBOL copybooks and building a metadata representation of it, the mechanics of doing this are menu-based rather than visual. Having said that, OpenLegacy enables rapid API front-ending of green screen applications through the use of built-in design themes and discovery wizards. But as usual with screen-scraping, this approach is best limited to simple transactions with few conversational menus.

	Comments
Mainframe-oriented service composition	Support covers 3270 or RPC access, or existing web services
Bottom-up and top-down service development	Both supported
Support for web services	Support for RESTful and SOAP-based web services
Minimal code generation	Minimal code generation
Automation facilities	Offers good automation of API Management tasks and also some service generation
Language support	All main mainframe languages supported
Support for additional mainframe resources	OpenLegacy supports RPC or 3270 access of mainframe applications, IMS Connect for IMS applications and any existing mainframe web services
Added-value orchestration	The Eclipse IDE enables orchestration of services in different back end systems
Ease-of-use	There is quite a bit of automation in the process, but the building of the metadata from the mainframe information still has technical elements
API Ecosystem support	Standards based security including OAuth and Java security. Also includes API monitoring and reporting
Testing tools	A number of test tools are provided that result in a powerful test harness for unit testing and continuous system testing
Governance and lifecycle support	Good support, including analytics for a more complete understanding of API usage
Mainframe experience	OpenLegacy relies on a standards-based, open source approach to mainframe API management, utilizing technologies such as RPCs. As a result, it is not a mainframe specialist company
API analytics support	Monitoring and analytics for a range of API usage metrics

Admin support for API ecosystem	As a generic Application Manager, broad API admin support is provided including role management and API management
Security	Standards-based approach includes OAuth and LDAP security
Monitoring and problem determination	Generally only provided at the generic level and not specifically within the mainframe
Integration with existing management framework	Standards-based
Bi-directional support	None
Choice of processing location	The Connectors run in the mainframe but the main work is done in a Java environment, located on premise, in the client or in the cloud.
Performance / scalability	The approach of packaging everything needed for the service in one micro-application should help to optimize performance for some workloads
Exploit native operating system high performance options	Nothing specific since the approach is very standards-based

Figure 12: OpenLegacy API Software assessment

Rocket Software

Rocket Software was founded in 1990. It has built a wide portfolio of solutions through acquisition and development. In 2006 Rocket acquired Seagull's BlueZone and LegaSuite integration products, providing terminal emulation and screen-scraping respectively. More recently, the LegaSuite functionality has been brought together with API management capabilities to become Rocket API. For the purposes of this study of mainframe API tools, Rocket API will be the focus product.

The diagram below shows where Rocket API fits in the API architecture as laid out in the Lustratus report "Best-of-Breed Mainframe API Enablement":

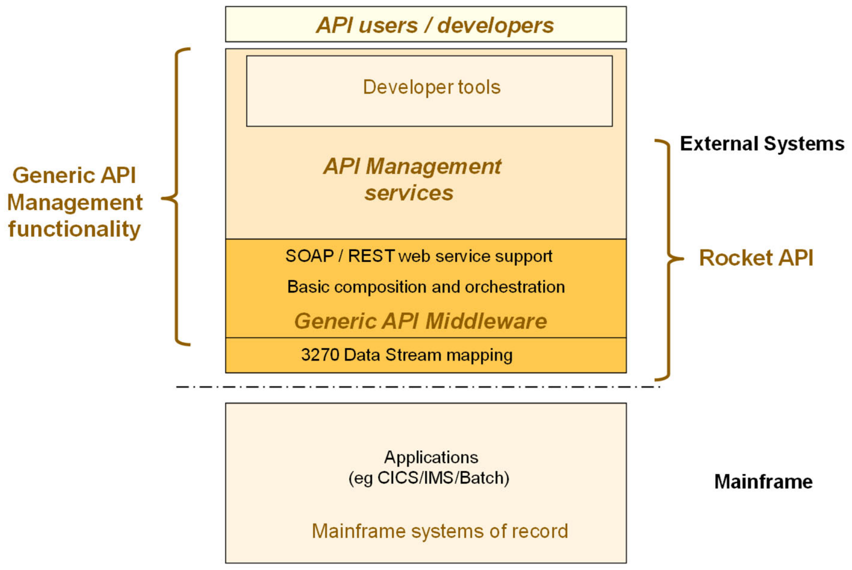

API users / developers
Developer tools
API Management services
SOAP / REST web service support
Basic composition and orchestration
Generic API Middleware
3270 Data Stream mapping
Generic API Management functionality
External Systems
Rocket API
Applications (eg CICS/IMS/Batch)
Mainframe systems of record
Mainframe

Figure 13: Rocket API's position in the mainframe API architecture

Rocket API

Rocket API provides a wide range of generic API Management services, although these are currently geared more from an enterprise point of view than from an API Economy one. As an example, at the time of our research Rocket API does not provide API documentation in Swagger form, although there are a number of user-driven initiatives to make this happen. There are also limited tools for managing third party developers. However, given the focus of this report is API enablement of IBM mainframes, the only mainframe-specific consideration with record is the provision of 3270 data stream mapping, or screen-scraping.

Screen-scraping is one of the oldest forms of mainframe integration, along with terminal emulation and file transfer. To put it simply, screen scraping is all about making the mainframe application think it is talking to a 3270 screen. When the local code wants to drive a mainframe application, the desired 3270 data stream is set up as if it had been typed into a mainframe green screen. Similarly, when the result is sent back to the green screen to display, the 3270 data stream is mapped into whatever makes sense to the calling environment.

The appeal is obvious; by building 3270 data streams to drive mainframe applications, the applications can be driven without any changes to the mainframe. In addition, the internals of those mainframe applications can change without any effect on external callers, unless the 3270 data streams change of course. However, there are serious drawbacks which tend to limit the applicability of this approach to certain specific scenarios. If the mainframe application is simple, for example a price lookup for a particular product number, then there is only one green screen to map and the call is a simple in/out to the mainframe. But CICS applications in particular are often far more conversational, with the user clicking options on a screen to then be carried to another screen and so on. The screen scraping approach requires not only the mapping between the 3270 data streams and the local user interface technology, but also a round trip to/from the mainframe for every step. As a result, screen scraping tends not to be scalable and remain rather inflexible and brittle.

Mainframe API functionality

Rocket API consists of a number of tools for supporting z/OS-based mainframe APIs. The Windows-based Rocket API Builder provides a wizard-like tool for mapping between green screen fields and data and the desired calling environment. There is also a tool for optimizing repetitive mappings, the API Flow Recorder. Once created, the API is deployed through the Rocket Access and Connectivity hub which can run on a variety of non-mainframe platforms including Windows, IBM , UNIX and Linux. At the time of this research, Rocket does not offer Swagger-based documentation of the APIs. It should also be noted that CICS support requires a Liberty server, meaning more moving parts to control.

Composite APIs can be constructed graphically with the API System Orchestration tool. It is worth mentioning that although not specifically part of the mainframe API scenario, Rocket API provides a range of monitoring and management tools to manage APIs in general. This includes role-based security, API user tracking and a prioritization mechanism.

So how does Rocket API match up to the Lustratus best-of-breed criteria? The Lustratus criteria are grouped into three sections, the first being Development and Deployment.

	Comments
Mainframe-oriented service composition	Support is limited to 3270 datastream access to mainframe applications, ideally single pass ones since conversational transactions will require a lot of comms resources travelling back and forth
Bottom-up and top-down service development	Both supported at a basic level but again restricted to using 3270 data streams
Support for web services	Support for RESTful and SOAP-based web services, although there is no Swagger publishing support at this time
Minimal code generation	No code required apart from the files controlling the 3270 datastream mapping
Automation facilities	Support to accelerate mapping of repetitive field structures
Language support	Since no code is required on the mainframe, there is no requirement for language support as long as 3270 screens are used
Support for additional mainframe resources	Rocket provide JDBC access to data, but application interaction is limited to those that use 3270 data streams
Added-value orchestration	Fairly standard orchestration provided, but all outside the mainframe
Ease-of-use	The mapping tools and lack of code help to make the 3270 mapping

	easier, but forcing the interaction to be at the screen interface means conversational transactions will quickly get complicated
API Ecosystem support	Good support for security, monitoring and general administration at the off-mainframe level, but nothing inside the mainframe
Testing tools	Basic testing tools provided
Governance and lifecycle support	Decoupling the caller from the mainframe enables mainframe transaction logic to change transparently, as long as they do not require screen changes
Mainframe experience	Rocket has grown primarily through acquisition and supports a wide range of platforms, but it has little z/OS mainframe skills
API analytics support	Rocket has a range of monitoring and analytics offerings, although of course these are all for the off-mainframe environment
Admin support for API ecosystem	Standard administration tools provided.
Security	Uses security features in the underlying mainframe systems
Monitoring and problem determination	Logging is through the underlying mainframe systems
Integration with existing management framework	Through underlying mainframe systems
Bi-directional support	None
Choice of processing location	Rocket API does not have any component residing on the mainframe. Processing is all off-host
Performance / scalability	Performance and scalability will suffer with more complex and more heavily conversational transactions
Exploit native operating system high performance options	Support through underlying mainframe systems, eg CICS, IMS

Figure 14: Rocket API assessment

Summary

This assessment looked at six solutions to mainframe API-enablement, making the potential selection task appear rather daunting. However, by first giving some thought to the aims of the mainframe API enablement project, the task can probably be simplified considerably. While there is no hard and fast rule, the general principles apply; if the aim is to provide access to a few specific mainframe query-style applications, with the longer term objective being to perhaps even replace the mainframe, then a simple approach such as screen scraping may well suffice. But if there is scope either now or in the future for a broader API enablement strategy to get the best possible value from the massive assets embodied in the mainframe, a mainframe-specific solution that is flexible and comprehensive is almost certainly the safest approach. For example, offering CICS application access may be all that is required today, but if in the future it is necessary to access other systems such as batch, IMS or third party packages then having to switch supplier will be costly and inefficient.

The other factor that must be taken into account is the availability of mainframe skills, both internally and within the chosen supplier. While a number of mainframe API enablement tools claim not to require any mainframe skills, the reality is that for anything other than a very simplistic application example there will almost certainly need to be some level of mainframe expertise and understanding required. It is not so important whether these skills are available internally or from the supplier, but it is important that they exist and are accessible to mitigate any risk.

The checklist tables should point the way to knowing what questions to ask while carrying out the vendor selection process, but they remain simple guidelines. In the end, companies must draw their own conclusions based on their own analysis.

About Lustratus Research

Lustratus Research, founded in 2006, aims to deliver independent and unbiased analysis of global software technology trends for senior IT and business unit management, shedding light on the latest developments and best practices and interpreting them into business value and impact. Lustratus analysts include some of the top thought leaders worldwide in infrastructure software.

Lustratus offers a unique structure of materials, consisting of three categories—Insights, Reports and Research. Insights offer concise analysis and opinion, while Reports offer more comprehensive breadth and depth. Research documents provide the results of practical investigations and experiences. Lustratus prides itself on bringing the technical and business aspects of technology and best practices together, in order to clearly address the business impacts. Each Lustratus document is graded based on its technical or business orientation, as a guide to readers.

Terms and Conditions

© 2018—Lustratus Research

Customers who have purchased this report individually or as part of a general access agreement, can freely copy and print this document for their internal use. Customers can also excerpt material from this document provided that they label the document as Proprietary and Confidential and add the following notice in the document: “Copyright © Lustratus Research. Used with the permission of the copyright holder”. Additional reproduction of this publication in any form without prior written permission is forbidden. For information on reproduction rights and allowed usage, email info@Lustratus.com.

While the information is based on best available resources, Lustratus Research disclaims all warranties as to the accuracy, completeness or adequacy of such information. Lustratus Research shall have no liability for errors, omissions or in adequacies in the information contained herein or for interpretations thereof. Opinions reflect judgment at the time and are subject to change. All trademarks appearing in this report are trademarks of their respective owners.

Steve Craggs trading as **Lustratus Research**
www.lustratus.com

Ref SC/LR/31675249V2.0