Cloud Platforms Integration Framework

A dissertation submitted for the Degree of Master of Information Technology

N. D. Kaduruwana

University of Colombo School of Computing

2019

Abstract

In this report, an attempt is made to develop an extensive, scalable middleware framework solution named Cloud Platform Integration Framework (CPIF) to port different cloud platforms which facilitate consumer applications to utilize cloud based resources from different cloud platforms as required. The ultimate goal of this solution is to reduce the development and long-term maintenance cost when an application needs to connect to multiple cloud platforms or either migrating to a different cloud platform.

Primary focus of CPIF solution is to implement a generic framework by utilizing the plug-in architecture pattern. It acts as a cloud platforms independent solution which provides a flexibility to implement cloud platform dependent communication channels as pluggable components with respective integration technologies. In future this framework facilitates an option of porting a new cloud technology by developing it as a new plug-in component.

CPIF solution consumes cloud based services by using generic set of interfaces where any application could directly integrate with it rather concerning about its integration mechanism. Therefore, in future, user should be able to switch into different cloud technologies with minimal configuration changes due to use of plug-in architecture pattern without any implementation changes.

In addition to that, CPIF solution deploys as a Windows Service. Therefore, it can be independently hosted in a different machine and completely decouple with the consumer application. Also it provides an opportunity to integrate any consumer application which is implemented using any other technologies (E.g. Java, PHP, etc).

CPIF solution is implemented using Microsoft .NET related technologies. (.NET Framework 4.5). CPIF solution implements the communication channels for the cloud based resources such as Microsoft Azure Queue and Blob storages and similarly Amazon Web Services Simple Queue Storage and S3 Bucket in order to efficiently transfer different sizes of data. Also CPIF solution can be easily extended to support any other cloud platform by following its plug-in component design.

This solution is strictly evaluated for Microsoft Azure cloud platform by executing wide range of test cases which includes functional and non-functional test cases. The evaluation results summaries the strength and weakness of the CPIF solution.

Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: N. D. Kaduruwana

Registration Number: 2016/MIT/027

Index Number: 16550272

_____

Signature: Date:

This is to certify that this thesis is based on the work of Mr. N.D. Kaduruwana under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Prof. N. D. Kodikara

_____

Signature: Date:

Acknowledgements

It is my pleasure to thank many individuals who contributed to the success of this project. First and foremost I thank to Prof. N.D.Kodikara, Senior Professor at the University of Colombo School of Computing (UCSC) for his valuable guidance and constructive advises given me as the project supervisor.

Secondary, I thank to all academic staff members of the University of Colombo School of Computing (UCSC) division for their kind support throughout this project.

Finally, I thank to my colleagues in MIT 2016 program for their generous support given me during this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	**Explanation**
CPIF	Cloud Platforms Integration Framework
API	Application Program Interface
AWS	Amazon Web Services
IAAS	Infrastructure As A Service
PAAS	Platform As A Service
SAAS	Software As A Service
MEF	Managed Extensibility Framework
CLR	Common Language Runtime
PE	Portable Executable
FIFO	First In, First Out
TCP	Transmission Control Protocol

Chapter 1 Introduction

As of today, the Cloud computing is one of the fast-growing technology where most of the companies around the globe are moving their software solutions into the Cloud environment due to the vast range of services it provides and also the benefits of having flexibility, disaster recovery, cost, performance and reliability.

Currently, there are many competitive cloud vendors who provide cloud based services by facilitating wide range of services along with technology enhancements. Top cloud vendors in the industry as of today are Microsoft, Amazon, Google, IBM and Oracle. Each of the cloud platform belongs to the respective vendors provides different benefits such as benefits over cost, usability, adaptability, etc.

However, due to the different underlying technologies used by different vendors, even though they provide similar services, their cloud platform integrations are totally different one to another. For an example, today if my company's software solution is integrated to the Microsoft Azure cloud platform and future due to a technology change decision, company decided to move to the Amazon Web Services cloud, it will require a significant effort for the change of the design and code refactoring and ultimately it will increase the cost the particular change request.

In addition to that, if a software solution requires to connect to two cloud platforms and use similar services (for example Microsoft Azure and WSO2 cloud platforms), still the software platform integrations required to be developed individually, which ultimately results to increase the effort, cost and moreover, difficulty of maintaining two different software integration solutions.

1.1 Scope of the Project

This project targets to implement a generic framework as a middleware solution (CPIF) by integrating multiple Cloud platforms. Therefore, any application which utilizes this framework can connect to the desired cloud platform as well as provide the option of changing cloud platforms with minimal configuration changes. The project scope includes:

- Implement a generic framework by utilizing Plug-in Architecture which should adhere to the 'Separate of Concern' design pattern
- CPIF framework will be developed by using Microsoft .NET technologies
- CPIF framework act as a middleware that can be integrated to any application which developed using any technology
- CPIF framework targets to implement generic set of interfaces for accessing File Storage, Message Queues in the desired Cloud platform
- The backend integrations of different Cloud platforms will be implemented as plug-ins where the generic framework should support deploying a plug-in with minimal set of configuration changes
- In this project scope, plug-ins will be developed to access Microsoft Azure cloud and Amazon Web Services (AWS) Cloud platforms

- CPIF framework should support any new plug-in developed in future for a different cloud platform
- Sample .NET application clients will be developed to showcase the middleware framework integration and data transfers with different Cloud platforms

1.2 Motivation

- Currently, many cloud based projects are getting developed in the industry due to rapid growth of technologies related to the cloud platforms. However, since the consumer applications are highly coupled with the respective cloud platform which is integrated, the flexibility of changing the cloud platform or reusability of existing codes to integrate to a different cloud platform is extremely poor.

  The proposed CPIF solution will provide a generalized framework, where developers can reduce the complexity of platform integrations. Also quality assurance team could reduce their testing effort due to integration already tested framework and make maintenance and support engineers' lives to be easier due to plug-in oriented framework. Ultimately, the CPIF solution will reduce the cost for the customer by giving opportunity of choosing any cloud platform as they desired.

- Proposed CPIF framework is to consume Cloud based services using generic set of interfaces where any application could directly integrate with the framework rather concerning about the Cloud technology. The major benefit here is complete development effort of new cloud platform integration can be cut-off and instead, support engineer should be able to switch into different cloud technologies with minimal configuration changes due to its Plug-in architecture.

- Implement a generic framework by utilizing the plug-in architecture. The vendor dependent cloud communication channels will be developed as plug-in. Therefore, in future the framework facilitates the option of porting a new cloud technology by developing it as a new plug-in. For an example if the client needs to utilize the features of WSO2 cloud platform tomorrow, still this framework support this and we could develop the integration as a unique plug-in and port it without impacting the changes to the core framework

- CPIF solution should support offline capabilities. Which means the CPIF solution should be host and executes independently therefore, it could all the data transfer communication will be done in asynchronous mode without maintaining a synchronous connectivity with consumer application. The ultimate objective is bottlenecks of consumer application should not flow into the CPIF middleware solution, vice-versa.

- CPIF solution should be able to integrate the any consumer application which is implemented using any technology (E.g. Java, PHP, etc). Therefore, integration of CPIF solution is technology independent.

1.3 Summary of Chapters

Chapter 2: Background

This chapter includes an up-to-date and comprehensive review of relevant literature. Also it demonstrates the awareness and understanding of the background literature of this topic.

Chapter 3: Methodology

This chapter describes the structure of the overall system, design methodologies, logical diagrams, proof of concepts and the implementation which includes important code snippets. Also it describes the design constraints of this research project and how those were addressed.

Chapter 5: Evaluation

This chapter provides the complete assessment of developed system's effectiveness, efficiency and also the user friendliness of the system and explains how project goal and objective achieved. Major functional and non-functional test cases and test results are specified within this chapter for the evidence.

Chapter 6: Conclusion

This chapter describes the summary of the project outcome, deficiencies and future extensibility of the project.

Chapter 2 Background

2.1 Introduction

With the vast development of information technologies in 21st century, we named this Era as Information Age. Due to that fact the information technology became a backbone of industrial development and economical growth around the globe. Moreover, all countries move toward the concept of globalization by strengthening their interaction between organizations and people worldwide. Ultimately the availability and sharing information worldwide became the key factor as of today regardless of the distance. Therefore, the trend of information technology moves toward innovating and inventing new information storage devices and information distribution technologies. Global technology leaders put some significant effort to provide new technology innovations to make the effort worth. Even though there are multiple solutions invented for storing and sharing information, the challenge is to achieve following factors.

- Flexibility
- Reliability
- Disaster recovery
- Work from anywhere
- Security
- Cost saving

Without achieving above factors, it is difficult for any organization to adapt for the information sharing platforms and technologies since their ultimate goal is to increase the sales by optimizing the efficiency of producing product and services by mitigating the risks. As a solution the cloud computing based technologies were introduced by achieving above factors and moreover providing adaptability where any organization could utilize it. Therefore, it is one of the most successful technologies in this era and it became the future trend of any organization to adapt it because of its benefits to improve the cash flows. It provides a centralize platform for the information by providing flexible way to share the information which is much more effective for the business, as well as for the humans that run it. With the information technology revolution, the following Cloud computing technologies taking place on top of all other similar technologies.

- IAAS – Infrastructure As A Service
- PAAS – Platform As A Service
- SAAS – Software As A Service

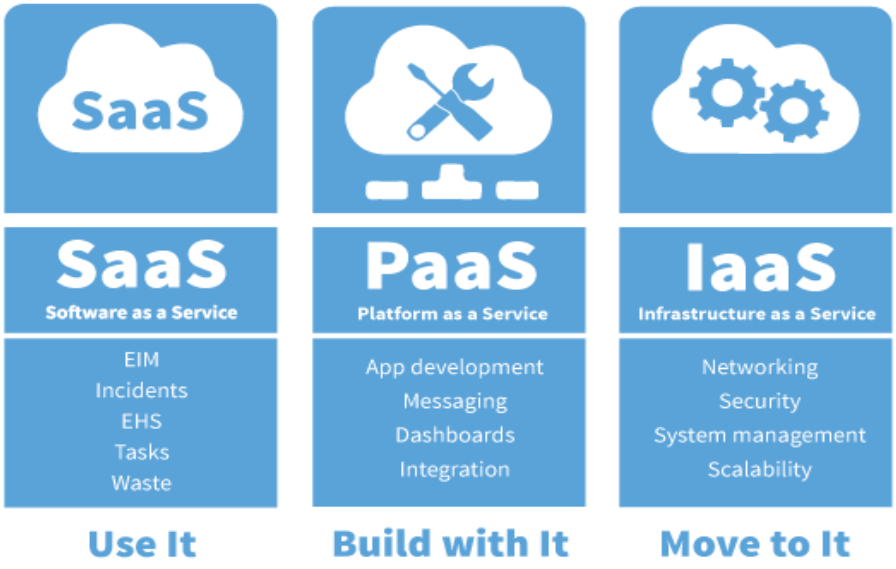

Figure 2.1 Cloud based services

Due to the increase of market value for the cloud data platforms, many vendors spent vast investment on building and strengthening their own cloud platforms. As of today there are many number of cloud platforms provided by different vendors in the industry and Microsoft, Amazon, Google, IBM and Oracle are the top vendors who gained high ranks due their ultimate features provided on their cloud platforms.

Following article provides the evolution of cloud platforms and importance of it.

[1]. Prof Dr, Claudia Müller-birn. (2012). Cloud Computing. Retrieved April 4, 2019 from http://citeseerx.ist.psu.edu/viewdoc/citations?doi=10.1.1.462.4311

[2]. University of California at Berkeley. (2009). Above the Clouds. Retrieved January 14, 2019 from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

Therefore, it is a costly decision to choose the right cloud platform since migrating one cloud platform to another will increase the cost due to the different integration patterns in addition to the vendor charges. Due to that reason either customer has to rely on one cloud platform even they need to migrate to another platform or either they have to migrate with an additional cost.

CPIF (Cloud Platform Integration Framework) provides a centralized middleware framework to solve the above problem where it facilitates customer to migrate one cloud platform to another only with configuration change and no development cost at all. In addition to that, it is beneficial for any company to utilize it due to its extensibility where the adaption of plug-in architecture pattern to the CPIF allows any new cloud platform to plug-into CPIF framework.

CPIF will solve the ongoing debate about having different cloud platforms with similar features but completely different integration technologies. Therefore, CPIF is a vendor independent middleware solution. It saves the time and development cost requires to switching between different cloud platforms.

Infrastructure as a Service (IaaS) provider is the complete cloud based solution to eliminate on-premises data centers, including servers, storage and networking hardware, etc. It supplies infrastructure components which include monitoring, log access, security, load balancing and clustering, as well as storage resiliency, such as backup, replication and recovery. CPIF utilizes the IaaS based services to interact with Cloud platforms for effective data communication. Therefore, it completely isolates the customer application from utilizing laaS services to build a data communication channel and CPIF will take the full responsibility of it. Therefore, by utilizing CPIF solution will cut-down the development cost required to integrating to a cloud platform and provide the flexibility of connecting to the preferred cloud platform by changing the CPIF configuration settings.

In this project scope, 2 major cloud platforms are chosen to build the CPIF middleware solution. Those two cloud platforms are Microsoft Azure and Amazon Web Services (AWS). Currently, those 2 cloud platforms are high ranked cloud solution providers due to their extensive features. Following table provides the IaaS based service comparison of top industry leading cloud platforms.

	Amazon AWS	**Microsoft Azure**	**Google Compute Engine**	**Rackspace**	**IBM Smart Cloud**	**HP Cloud**
Virtual CPUs	43108	43108	43108	43108	43116	43108
Memory	613MB - 68.4GB	1.7GB - 14GB	3.7GB - 52GB	512MB - 30GB	2GB - 16GB	1GB - 32GB
Cost/HR	Free - $4.60	$0.02 - $2.04	$0.145 - $1.375	$0.022 - $1.20 (Linux)	See cost IBM estimator	$0.035 - $1.12 (Linux)
Storage Costs	$0.095 GB/mo (S3)	0.095	$0.10 GB/mo	$0.15 GB/month	See cost IBM estimator	$0.10 GB/mo
Availability	0.9995	0.9995	0.9995	0.9999	0.999	0.9995
Compensation for Outage	10% - 30% credit	10% - 25% credit	10% - 50% credit	5% off server fees	Credit equal to outage time	5% - 30% credit
Distinguishing Features	Wide array of storages & specialized services.	Flexibility in administrative control.	Designed for data intensive & high performance	Easy to use.	Easy to manage multiple developers, IBM assets.	Easy access to HP's CDN.

Table 2.1 Cloud platforms feature comparisons

2.2 Microsoft Azure Cloud Platform

Microsoft Azure provides a full range of solutions for the developers to build enterprise applications. The Azure platform itself provides the automatic application deployment and managing virtual machines as scale. User could on-demand scale the resources at any capacity as required within few minutes of time. It provides the ideal platform to develop enterprise applications which requires high load of data storage and on-demand scaling and also many other non functional requirements.

Key Strengths:

- Unlike AWS, Azure has a unique focus on hybrid cloud setups (combination of private and public cloud services). As of today it is the common cloud architecture for large enterprises to have hybrid cloud setups (setups that often can't be easily achieved with AWS).
- Also best platform to utilize Microsoft Services (SQL, Active Directory, and .NET) and it will integrate quickly and easily with existing on-premises Microsoft infrastructure.
- Provides customer the wide range of options like 'Basic' service level, which is a bargain-priced compute service that does not include auto-scaling or load balancing, making it an ideal selection for things like development environments and other non-public-facing sites.
- Azure's unique strengths, enhancements and moreover the customer satisfaction predicted that it will be the largest IaaS provider by 2019.

Weaknesses:

- Lack of Hyper-V Snapshot Support
- Inability to Upload Custom Images
- Provisioning Virtual Machines in the Cloud Takes Longer than On-Premise
- Lack of Integrated Backup
- Poor Management GUI and Tools

2.3 Amazon Web Services Cloud Platform

Amazon Web Services became more popular due to the broad variety of infrastructure applications and flexible platforms. It allows easy and flexible ways for users to access computing power, data storage and it core feature as necessary for the developers. Also it provides a vast range of developer tools, management tools, mobile services and applications services.

Key Strengths

- AWS was the first to offer public IaaS, as far back as 2006 and due to that reason it key strengths is the maturity of its IaaS offerings
- AWS has vast range of services available, from DNS routing to caching to load balancers: virtually anything you could ever need out of the cloud, AWS can deliver.

- AWS has the most data centers of any IaaS provider, which means they have the most comprehensive global coverage and the most robust, reliable network.
- These two key strengths are often the main draw to AWS for many customers: it's almost guaranteed that you can do what you need to do on AWS, no matter how obscure, and it will offer suitable reliability for even the most sensitive applications.

Weaknesses

- AWS is actually pretty expensive as compared to Azure and most of other Cloud vendors.
- The platform involves with complexities and quite steep learning curve and mastering all the required services on your own is hardly possible. For example, to build a truly resilient, secure and fail-safe IT infrastructure the company would have to figure out the use of routing and IAM politics - or hire an AWS-certified partner to configure the thing for them.

Chapter 3 Methodology

3.1 Introduction

This chapter focuses on describing methodologies used for designing of Cloud Platforms Integration Framework. Since this is a middleware solution, the design of this framework should achieve the general cloud platform data transfer functional requirements as well as non functional requirements by adhering to the software design patterns and methodologies.

In addition to the design of core framework in CPIF solution, mainly this chapter provides the evidence and examples of designing one of the plug-in components which is similar of designing other plug-in components.

As a summary this chapter covers the design methodologies of following CPIF application components.

- **CPIF core framework** –All the plug-in components relates to different cloud platforms are connecting to the core framework. Therefore, the design of core framework should accommodate the future extensibility requirement of supporting new plug-in components associate with new cloud platform. Also core framework facilitates configuring plug-ins, and provides simplified common interface to the other applications to integrate and use CPIF middleware solution
- **Plug-in component design –** This section covers the generic plug-in component design independent of the cloud platform technology. However, Microsoft Azure cloud platform is selected to provide the evidence and examples of designing one of the plug-in components.

3.2 Design of CPIF Core Framework

The design of CPIF Core Framework should mainly achieve the following 2 requirements

1. Configuring Plug-in components without any core framework implementation changes
2. Provides common interface for connecting application to utilize the CPIF solution

The CPIF middleware solution will be implemented using Microsoft .NET related technologies. In order to support Plug-In components, the "Plug-in Architecture" is used. Main, goal of utilizing Plug-in Architecture is to achieve the "Separation of Concern" design pattern. The Separation of Concern design pattern is one of the most useful design patterns, which helps to develop loosely coupled system which divides the responsibility in different units. Primary purpose of this design is one unit completely responsible of undertaking the responsibility and will not take any other responsibilities. Next section further elaborates how plug-in component design adheres to this design pattern.

When implementing CPIF core framework using plug-in architecture, following Microsoft technologies have been deeply evaluated.

- .NET reflection
- Managed Extensibility Framework(MEF)

.NET Reflection

.NET reflection is the mechanism of dynamically loading assemblies during .NET runtime. Basically, during the .Net compile time metadata created with Microsoft Intermediate language (MSIL) are getting stored in the Manifest file. Both Metadata and Microsoft Intermediate Language relates to the code we developed together wrapped in a Portable Executable (PE).

The reflection is used to dynamically access the PE file at the .NET runtime and read metadata, then creates the respective Types associate with the PE file and finally, invoke the methods and access properties inside the PE file.

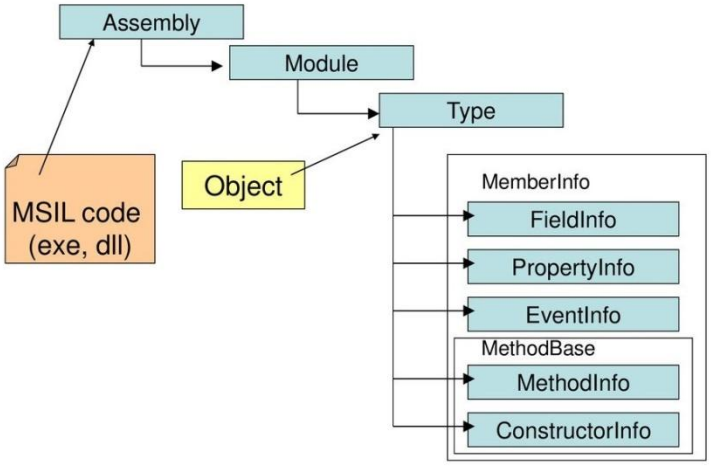

Figure 3.1 Logical hierarchy of .NET reflection

The reflection provides the flexibility and expansibility for your application by completely omitting the coupling to improve the application adaptability. Therefore, it would be one of the good choices to achieve modularization and to develop the plug-in based applications.

Managed Extensibility Framework(MEF)

MEF is introduced with .NET 4.0 framework and it is built on top of the Reflection API. It is specially design to modularizing your application and to support extensibility. The Reflection API has a strict constraint where you need to configure your loading assembly namespaces in order to load it dynamically at the runtime. Also does not provide a facility to tag your loading assemblies. Therefore, it has the limitation where other application might also have the opportunity of using the pluggable components.

However, the MEF addressed all the above limitations. Basically, MEF allows tagging pluggable components with additional metadata which facilitates querying and filtering at the runtime. It provides standard host for any plug-in without the need of configuring your assembly

namespaces. Therefore, it is as self-container for any plug-in which contains the metadata information which could use at the runtime.

In the scope of CPIF solution, choosing MEF has a huge advantage. Since CPIF solution requires a flexible plug-in based solution, the MEF is the ideal technology selection. Moreover, the attribute based discovery mechanism in MEF will promote extensibility for CPIF plug-ins without the need of any additional configuration. It has the option of categorizing the classes related to plug-in components using meta-data attributes without the need of registering them individually.

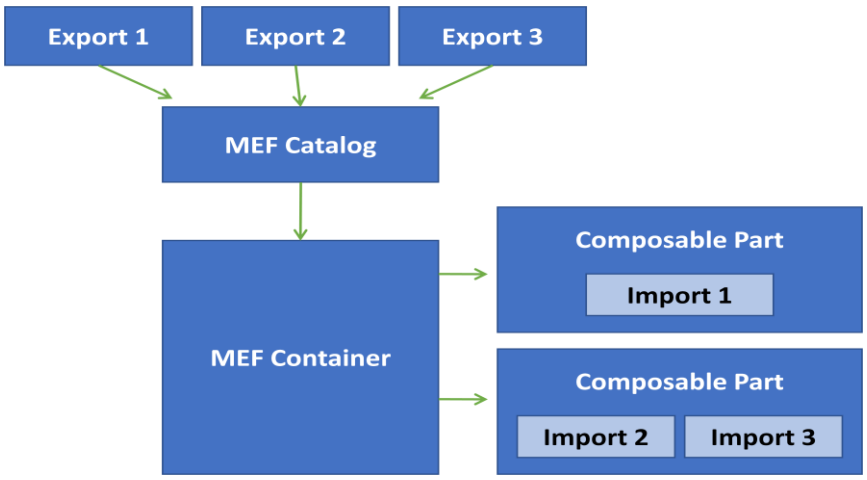

Figure 3.2 MEF framework components

As per the above diagram, "Exports" are the plug-in components which are decorated with exported values of plug-in components are all stored in a container. Following code snippet indicates how CPIF plug-in components are decorated with Export metadata data values.

```
[Export(typeof(IOperation))]
[ExportMetadata("Command", "AzureCloudReceiveProcessor")]
0 references
public class AzureCloudReceiveProcessor : IOperation
{
```

As explained in the above diagram, all the plug-in components are accessed through the "Catalog". All the exported data are getting stored in a container.

The following class shows how CPIF framework built a custom MEF container and stores inside it all the plug-in exports from the directory where the plug-in components assemblies reside.

```
0 references
private Mef_Container()
{
    var agregateCatalog = new AggregateCatalog();
    agregateCatalog.Catalogs.Add(new AssemblyCatalog(typeof(Mef_Container).Assembly));
    agregateCatalog.Catalogs.Add(new DirectoryCatalog("plugins"));

    _compositionContainer = new CompositionContainer(agregateCatalog);

    try
    {
        this._compositionContainer.ComposeParts(this);
    }
    catch (CompositionException compositionException)
    {
        Console.WriteLine(compositionException.ToString());
    }
}
```

During the composition of an application to use the CPIF framework, the exports in the catalog are assigned to requested imports. Following is the code snippet associate with imports of common interface in order to use the plug-in component features.

```
class Program
{
    [Import(typeof(ICore))]
    public ICore core;
```

3.3 Plug-in component design

In this scope of project CPIF framework contains two completely developed and tested plug-in components. Those two plug-in components contain individual implementations to facilitate the data exchanges of Microsoft Azure and Amazon AWS cloud platforms. In addition to that, CPIF framework supports extensibility by providing opportunity to develop new plug-in component with completely different cloud platform technology and plug-in to the CPIF framework with minimal configuration changes keep the new cloud platform related connectivity information however, no code changes required.

Since the any .NET based consumer application could integrate with CPIF framework, all the plug-in components must be well designed and implemented to cater all the data exchange requirements. Following section explains the design constraints respective to the user requirements which have been followed when designing CPIF solution.

3.3.1 Design Constraints

1. User should be able to exchange data with preferred cloud platform without any size limitation using CPIF middleware solution. Plug-in component should be completely responsible of handling large size of messages.
2. Plug-in component should support offline data communication capabilities.
3. CPIF should support high data loads and delivered respective to the network bandwidth and available hardware computing resources. CPIF solution should not have any performance bottlenecks when supporting high data loads
4. Design of plug-in component should support a loosely coupled design by following "Separation of Concern" design pattern. Therefore, same design could be followed when implementing new plug-in components
5. Any exception occurs within plug-in component should be properly handled within plug-in component itself and notify the calling consumer application with proper error details.

3.3.2 Addressing Design Constraints

Addressing Design Constraint -1

To handle the first design constraint, plug-in component is designed in a way that it could make decision when there is large size of messages.

For an example consumer application required to transfer following size of messages

Message No.	**Size**
Message -1	5 KB
Message-2	10 MB
Message-3	100 MB
Message-4	1 GB

Table 3.1 Message sample dataset

The design of plug-in component to cater the above message size requirement is completely depend on the data storage capacity supports by respective cloud platform

Please find below the data capacity limits of Azure cloud platform.

Azure Queue storage scale targets

Resource	**Target**
Max size of single queue	500 TiB
Max size of a message in a queue	64 KiB
Max number of stored access policies per queue	5
Maximum request rate per storage account	20,000 messages per second assuming 1 KiB message size

Resource	Target
Target throughput for single queue (1 KiB messages)	Up to 2000 messages per second

Table 3.2 Azure Queue storage scale targets

Azure Blob storage scale targets

Resource	Target
Max size of single blob container	Same as max storage account capacity
Max number of blocks in a block blob or append blob	50,000 blocks
Max size of a block in a block blob	100 MiB
Max size of a block blob	50,000 X 100 MiB (approx. 4.75 TiB)
Max size of a block in an append blob	4 MiB
Max size of an append blob	50,000 x 4 MiB (approx. 195 GiB)
Max size of a page blob	8 TiB
Max number of stored access policies per blob container	5
Target throughput for single blob	Up to 60 MiB per second, or up to 500 requests per second

Table 3.3 Azure blob storage scale targets

Reference:

Microsoft Azure. (2019). Azure Storage scalability and performance targets for storage accounts. Retrieved April 4, 2019 from https://docs.microsoft.com/en-us/azure/storage/common/storage-scalability-targets

Usually, Azure queue is used to store small file sizes and Azure blob is used to handle large file sizes. However, as per the above figures, both have limitations. Given the fact that Azure queue storage supports max size of 64 KB and blob storage supports max size of 100 MB for a single message.

Following flowchart describes the way of handling small and large file sizes the respective to the storage limitations

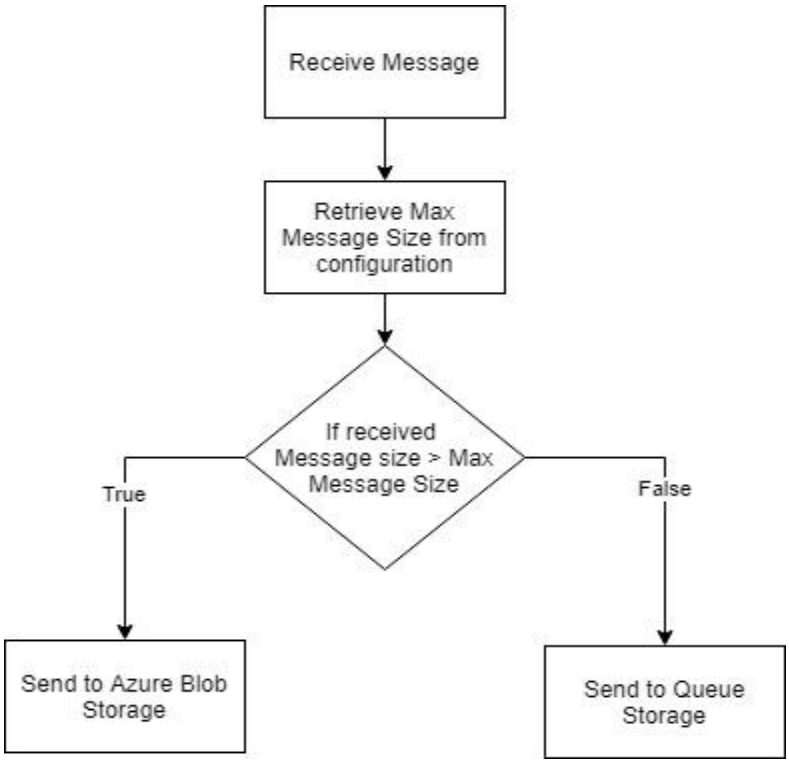

Figure 3.3 Flow chart for large message handling

Above diagram explains the how the plug-in component is designed to handle the small and large messages. The max size of the message is stored in the application configuration file where user could modify as required. For the small message sizes, still the queue storage is used, since queue storage provides efficient data transfers and wide range of features.

Therefore, as per the above design following table explain the how following file sizes are getting handled.

Message No.	**Size**	**Message Handling Mechanism**
Message -1	5 KB	Use message queue since max message size queue supports is 64 KB
Message-2	10 MB	Use blob
Message-3	100 MB	Use blob
Message-4	1 GB	Cannot use blob since max message size blob supports is 100MB. Therefore, this requires a special mechanism to disassemble the file size into smaller set of files and uploads to the blob. When receiving assemble the message accordingly. However, in this project scope, this will not be implemented. Therefore, max file size will be limited to 100MB size.

Table 3.4 Message handling mechanisms

Notes:

- Azure Service Bus Queue is utilized to develop the CPIF Azure based plug-in component. Please refer Appendix Sec. “6.1 Azure Storage queues and Service Bus queues” for more information.
- Additionally, the Azure Block Blob is utilized to develop the CPIF Azure based plug-in component. Please refer Appendix Sec. “6.2.Azure Block Blob vs Page Blob vs Append Blob” for more information.

Addressing Design Constraint -2

As per the 2nd design constraint, the plug-in component should support the capability of offline messaging.

The reason behind this constraint is, the message generation rate by the consumer application and the message delivering rate by the CPIF framework to the cloud platform could different. In that case, the consumer application should not get blocked or slowdown the message generation performance due to rate of message delivery supports by the CPIF framework.

On the other hand, message receiving rate by the CPIF framework from cloud platform and the message consuming rate by the consumer application could different. Therefore, CPIF framework should not slowdown its performance or gets blocked due to bottlenecks in the consumer application.

Another reason is, exception of consumer application should not bubble-up to the CPIF framework and vice-versa.

Due to the above-mentioned facts, the CPIF framework is implemented to support offline capabilities as represent in below diagram.

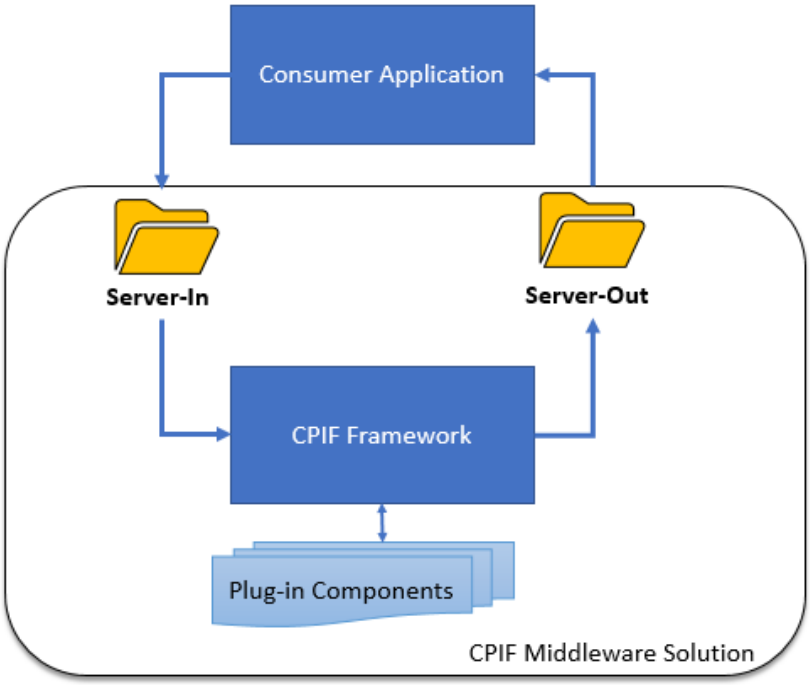

Figure 3.4 CPIF middleware offline connectivity

As explained in above diagram, data communication happened through preconfigured folders. “Server-In” folder is to receive files to the CPIF framework and “Server-Out” folder is to send files to the consumer application. According to this design, there is no direct data communication link between CPIF framework and the consumer application. Also, another advantage of this design is, the CPIF framework could completely execute as a separate job through a Windows Service or any other application hosting mechanism. Therefore, the CPIF middleware solution completely could completely execute as a completely isolated job rather coupling with consumer application. In this mechanism, the CPIF solution adhere to the “Separate of Concerns” design patter with a loosely coupled design.

Addressing Design Constraint -3

Since CPIF middleware solution works as an independent tool, it should support high data loads and guaranteed delivery.

In this case, the design should support low cohesion by reducing the complexity of delivering and receiving data loads. However, the actual performance could be measured during solution evaluation step by executing a performance test.

When dealing with Azure Service Bus queues, the Broker Message should not be expired due to delay of processing time. That will reduce the CPIF solution reliability due to the issue of guaranteed delivery. Typically, a Broker Message received through an Azure Service Bus queue has a strict expiration time. (maximum 5 minutes).

Therefore, in order to overcome this issue, after receiving the Broker Message from Service Bus queue, it has to be renewed during the processing time as specified below.

```
var stopwatch = new Stopwatch();
stopWatch.Start();
for (inti=0;i<=maxLimit;i++)
{
        if(stopWatch.Elapsed.TotalSeconds > 120)
        {
            Trace.TraceInformation("Renewing BrokeredMessage Lock");
            brokeredMessage.RenewLock();
            stopWatch.Restart ();
        }
}
```

Reference:

DotNet Artisan Cloud Consultant. (2016). Renew lock time for BrokeredMessage. Retrieved February 4, 2019 from http://dotnetartisan.in/avoiding-messagelocklostexception-using-auto-renew-pattern-for-brokeredmessage-service-bus-queue/

Addressing Design Constraint -4

As explained in above sections, the rule of thumb of designing CPIF middleware solution is to come-up with a loosely coupled design by adhering to the "Separation of Concern" design pattern.

Following diagram provides a high-level overall view of how Plug-in architectures is utilized to achieve the requirement of CPIF solution with a loosely coupled design

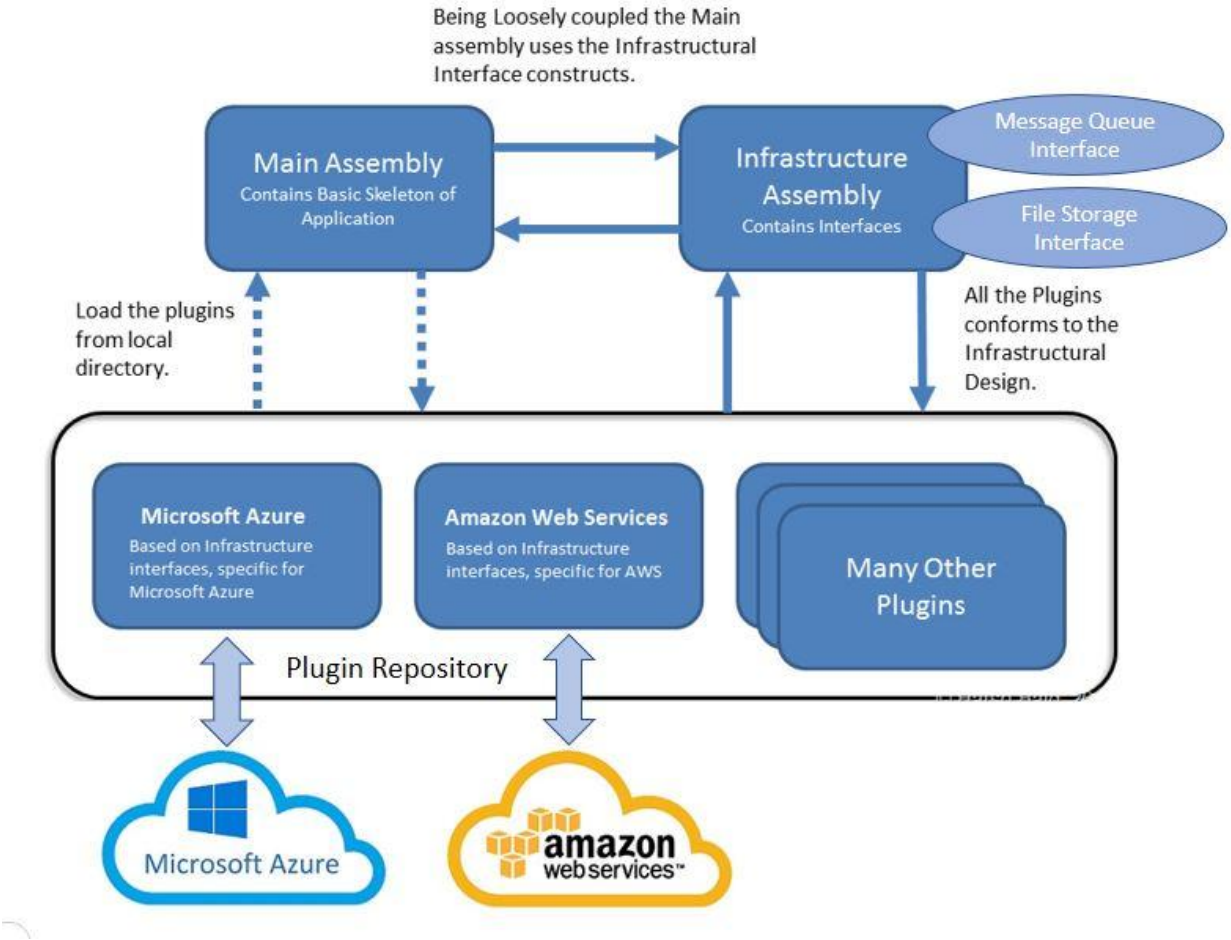

Figure 3.5 CPIF middleware solution components

The components of above architecture diagram are described below.

Main Assembly – This component contains the core of the CPIF architecture. The MEF framework is implemented within this assembly with the mechanism of loading plug-in components. Therefore, this component responsible for activating/deactivating plug-ins as needed. Refer "3.1 Design of CPIF Core Framework" section for more information

Infrastructure Assembly – This component contains all the interfaces to the external applications. Mainly, the Message Queue and File Storage interfaces are available within this component. Those interfaces are common and do not specific to any of the cloud platform

Microsoft Azure Plug-in – This is one of the plug-in in the framework which used to connect to the Microsoft Azure. The Microsoft Azure platform dependent logic resides within this component.
Refer "3.2 Plug-in Component Design" section for more information

Amazon Web Services Plug-in – This is one of the plug-in in the framework which used to connect to the Amazon Web Services(AWS). The AWS platform dependent logic resides within this comment.

Refer "3.2 Plug-in Component Design" section for more information

Addressing Design Constraint -5

Any exception occurs within plug-in component should be properly handled within plug-in component itself and notify the calling consumer application with proper error details. In order to achieve this requirement, a logging framework is introduced for the CPIF solution**.** The purpose of the logging framework is to provide logging capabilities to other components and business applications for debugging and tracing purposes.

The logging framework is implemented over the Log4net log engine. The logging framework is designed as a framework and is accessible from other application domains.

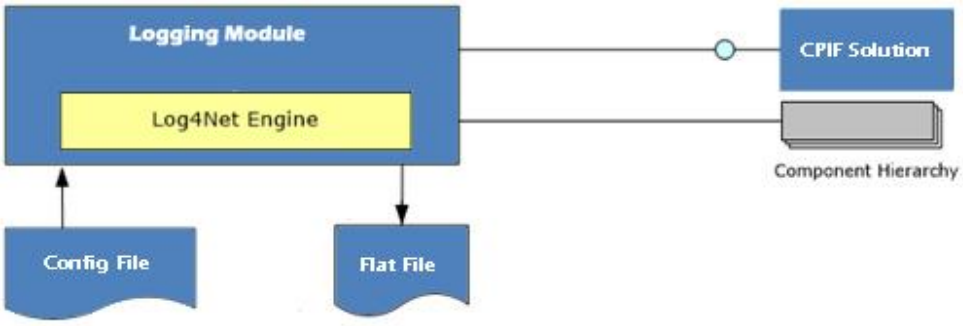

Figure 3.6 Logging framework components

The Log4net engine is a simple and an elegant way to log in errors, informational messages, and warnings.

3.4 Alternative Solutions

As per the current requirement, this application has been implemented using .NET Framework version 4.5 and Visual Studio 2015 version. This limit deployment environment only support Microsoft Windows platform with .NET CLR.

However, implementing this solution with the latest .NET Core technology will provide the benefit of supporting cross-platforms. Currently, the .NET Core delivers a fast and modular platform for creating server apps that run on multiple platforms such as Windows, Linux, and macOS. Additionally, it supports different types CPU architectures.

Chapter 4 Evaluation

4.1 Introduction

Since CPIF solution is a middleware solution which could be integrated into any cloud based application, this solution required to be deeply evaluated. As the outcome of the evaluation, the CPIF solution should achieve the high usability, reliability, performance and guaranteed delivery. Following table indicates how above factors are evaluated with respect to the CPIF testing.

Usability	CPIF middleware solution is a framework which contains plug and play logical modules. In fact that each plug-in component is designed to integrate seamlessly to the framework with minimal set of configuration changes. In the scope of usability testing, tester required to deploy the CPIF solution with consumer application by simulating a production environment and then deploy a new plug-in component by modifying exiting configurations and verify the functionality of new plug-in component.
Reliability	In the scope of reliability evaluation, all the functional test cases required to be executed to verify there no error occurred during the functional test case execution. All the functional flows required to be verified and during the scope of testing. Since this is a middleware solution, error and info logs required to be strictly monitored.
Performance	One of the core factors considered during the CPIF solution design is the performance. Sine CPIF solution includes a data delivery channel integrate with cloud platform, there should not be any sort of performance overhead due to the design of this solution. In the scope of performance testing, high, medium and low data loads required to be transferred with respect to the different file sizes and evaluate the performance.
Guaranteed Delivery	Since CPIF solution consists with data delivery mechanism from on-premises to cloud platform and vice-versa, there should not be any data-loss happened. More importantly, there should not be any data corruption during the delivery. This will be covered during functional and performance testing.

Table 4.1 Non functional requirements

4.2 Usability Testing

Following test cases are executed to evaluate the usability aspects of the CPIF solution. One of the CPIF core framework features of utilizing plug-in architecture pattern to facilitates application administrators to switching between different cloud platforms are evaluated by executing following test cases.

Test Case Field	Description
Test case ID	UT001
Test Severity	High
Name or Test Title	Switching to a different Cloud Platform - Uplink Testing
Description/Summary of Test	This test case is to validate the file upload feature of CPIF framework by switching to a different cloud platform after the CPIF is on production
Pre-condition	1. Two plug-in components are developed and ready for this test. Plug-in-1 is for Azure cloud platform and plug-in-2 is for AWS cloud platform 2. CPIF solution should be deployed and up and running 3. Plug-in component for Azure cloud platform should be deployed and running
Test Steps	1. Stop the server of CPIF solution 2. Copy and paste AWS Plug-in component into "Plug-in" folder locates under deployment folder 3. Modify the App.config file to add AWS connection related details 4. Start the server of CPIF solution 5. Copy and paste 10 files less than 100Kb and 10 files greater than 100Kb to the "Server-In" folder.
Test Data	10 files less than 100 Kb 10 files greater than 100 Kb
Expected Results	Files less than 100 Kb should be moved to the AWS queue Files greater than 100 Kb should be moved to the AWS bucket
Post-Condition	AWS queue and bucket should consist with transferred data
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.2 Usability testing test case 1

Test Case Field	Description
Test case ID	UT002
Test Severity	High
Name or Test Title	Switching to a different Cloud Platform - Downlink Testing
Description/Summary of Test	This test case is to validate the file download feature of CPIF framework by switching to a different cloud platform after the CPIF is on production
Pre-condition	1. Two plug-in components are developed and ready for this test. Plug-in-1 is for Azure cloud platform and plug-in-2 is for AWS cloud platform 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Files should be available at AWS queue and bucket
Test Steps	1. Stop the server of CPIF solution 2. Copy and past AWS Plug-in component into "Plug-in" folder locates under deployment folder 3. Modify the App.config file to add AWS connection related details 4. Start the server of CPIF solution
Test Data	10 files less than 100 Kb 10 files greater than 100 Kb
Expected Results	Files less than 100 Kb should be downloaded to the "ServerOut" folder from the AWS queue Files greater than 100 Kb should be downloaded to the "ServerOut" folder from AWS bucket
Post-Condition	AWS queue and bucket should be get cleared
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.3 Usability testing test case 2

Test Case Field	Description
Test case ID	UT003
Test Severity	High
Name or Test Title	Switching to a different Cloud Platform - Negative scenario
Description/Summary of Test	This test case is to validate the CPIF framework behaviour when there is no plug-in component deployed
Pre-condition	1. One plug-in (Azure) component should be developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plug-in component for Azure cloud platform should be deployed and running
Test Steps	1. Stop the server of CPIF solution 2. Remove all plug-in components from "Plug-in" folder locates under deployment folder 4. Start the server of CPIF solution
Test Data	N/A
Expected Results	CPIF framework should log and error in the error log file
Post-Condition	CPIF solution should be up and running and not be crashed
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.4 Usability testing test case 3

4.3 Reliability Testing

Under the scope of reliability testing following functional test cases are executed to verify the CPIF functionalities. During the scope of testing, major main functional flows will be tested to confirm the CPIF reliability aspects and the guaranteed delivery.

Note: Functional testing scope includes, testing of the framework and the Azure plug-in components. In the following section includes only the severity high test cases.

Test Case Field	**Description**
Test case ID	RT001
Test Severity	High
Name or Test Title	Azure Queue uplink Testing
Description/Summary of Test	This test case is to validate the file upload feature of CPIF framework using Azure Plug-in component for Azure Queues
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plug-in component for Azure cloud platform should be deployed and running 4. Azure service bus queue should be created in the Azure portal
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Make sure zero record count in selected queue 5. Copy and paste 10 files less than 100Kb to the "Server-In" folder. 6. Monitor queue count and the CPIF log files
Test Data	10 files less than 100 Kb
Expected Results	10 Files less than 100 Kb should be moved to the Azure service bus queue. Queue count should be 20. Which includes metadata files and the original files No error logged in under CPIF error log file Info log file should contains file names and delivery status to the Azure queue
Post-Condition	Azure service bus queue should consist with transferred data (20 files => 10 original Files + 10 metadata file) ServerIn folder should be empty
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.5 Reliability testing test case 1

Test Case Field	Description
Test case ID	RT002
Test Severity	High
Name or Test Title	Azure Queue Downlink Testing
Description/Summary of Test	This test case is to validate the file dowload feature of CPIF framework using Azure Plug-in component (Azure Queues)
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Azure servicebus queue should be created in the Azure portal 5. Azure servicebus queue should contains 20 files which includes 10 original files + 10 metadata files belongs to original files
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Make sure record count is 20 in the selected queue 5. Start CPIF server and monitor "Server-Out" folder. 6. Monitor queue count and the CPIF log files
Test Data	10 files less than 100 Kb
Expected Results	Files less than 100 Kb should be moved from Azure service bus queue to the "ServerOut" folder. 10 files should be created and queue count should be zero No error logged under CPIF error log file Info log file should contains file names and data consume status from the Azure queue
Post-Condition	Azure servicebus queue record count should be zero ServerOut folder should contains 10 files
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.6 Reliability testing test case 2

Test Case Field	Description
Test case ID	RT 003
Test Severity	High
Name or Test Title	Azure Blob uplink Testing
Description/Summary of Test	This test case is to validate the file upload feature of CPIF framework using Azure Plug-in component (Azure Blobs)
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Azure Queue and Blob should be created in the Azure portal
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Open a different browser and follow first two steps and navigate to the Azure Blob 5. Make sure record count is zero in the selected queue 6. Make sure record count is zero in the Azure Blob 5. Copy and paste 10 files less than 100Kb to the "Server-In" folder. 6. Monitor Blob count and the CPIF log files
Test Data	10 files greater than 100 Kb
Expected Results	Files greater than 100 Kb should be moved to the Azure Blob. Also Azure queue count should be 10, which includes metadata files belong to original files exist in Azure Blob No error logged under CPIF error log file Info log file should contain file names and delivery status to the Azure Blob
Post-Condition	Azure Blob should consist with transferred 10 files Azure servicebus queue should consist with 10 metadata files ServerIn folder should be empty
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.7 Reliability testing test case 3

Test Case Field	Description
Test case ID	RT 004
Test Severity	High
Name or Test Title	Azure Blob Downlink Testing
Description/Summary of Test	This test case is to validate the file download feature of CPIF framework using Azure Plug-in component. (Azure Blobs)
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Azure Blob should be created in the Azure portal 5. Azure Blob should contains 10 files 6. Azure Queue should contains 10 metadata files belongs to original files which are in blob
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Open a different browser and follow first two steps and navigate to the Azure Blob 5. Make sure record count is 10 in the selected queue 6. Make sure record count is 10 in the Azure Blob 7. Start CPIF server and monitor "Server-Out" folder. 8. Monitor queue count and the CPIF log files
Test Data	10 files greater than 100 Kb
Expected Results	Files greater than 100 Kb should be moved from Azure Blob to the "ServerOut" folder. 10 files should be created under particular folder. The Azure Blob count and the Azure queue count should be zero No error logged in under CPIF error log file Info log file should contains file names and data consume status from the Azure Blob
Post-Condition	Azure Blob count and the Azure queue count should be zero ServerOut folder should contain 10 files
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A
Test Case Field	Description

Table 4.8 Reliability testing test case 4

Test Case Field	Description
Test case ID	RT 005
Test Severity	High
Name or Test Title	Azure Queue and Blob Downlink Testing in Parallel
Description/Summary of Test	This test case is to validate the file download feature of CPIF framework using Azure Plug-in component simultaneously through Azure Blob and Queue.
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Azure Blob should be created in the Azure portal. Azure Blob should contains 10 files 6. Azure Queue should be created in the Azure portal and it should contains 30 messages (10 original messages + 20 metadata files)
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Open a different browser and follow first two steps and navigate to the Azure Blob 5. Make sure record count is 20 in the selected queue 6. Make sure record count is 10 in the Azure Blob 7. Start CPIF server and monitor "Server-Out" folder. 8. Monitor blob and queue counts and the CPIF log files
Test Data	10 files less than 100 Kb 10 files greater than 100 Kb
Expected Results	Files greater than 100 Kb should be moved from Azure Blob to the "ServerOut" folder. Files less than 100 Kb should be moved from Azure Queue to the "ServerOut" folder. Total 20 files should be created under particular folder. The Azure Blob count and the Azure queue count should be zero No error logged in under CPIF error log file Info log file should contains file names and data consume status from the Azure Blob and Queue
Post-Condition	Azure Blob count and the Azure queue count should be zero
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.9 Reliability testing test case 5

Test Case Field	Description
Test case ID	RT 006
Test Severity	High
Name or Test Title	Azure Queue and Blob Uplink and Downlink Testing in Parallel
Description/Summary of Test	This test case is to validate the file upload and download features of CPIF framework using Azure Plug-in component simultaneously through Azure Blob and Queue.
Pre-condition	1. Azure plug-in component is developed and ready for this test. 2. CPIF solution should be deployed and up and running 3. Plugin component for Azure cloud platform should be deployed and running 4. Azure Blob should be created in the Azure portal. Azure Blob should contains 10 files 6. Azure Queue should be created in the Azure portal and it should contains 30 messages (10 original messages + 20 metadata files)
Test Steps	1. Browse https://portal.azure.com in web browser 2. Login to Azure portal by providing authentication details 3. Navigate to All Resources > Service Bus > Select queue and open queue overview page 4. Open a different browser and follow first two steps and navigate to the Azure Blob 5. Make sure record count is 20 in the selected queue 6. Make sure record count is 10 in the Azure Blob 7. Copy 20 files to "Server-In" folder. (10 files > 100Kb and 10 files <100 Kb) 7. Start CPIF server and monitor "Server-Out" folder. 8. Monitor queue count and the CPIF log files
Test Data	10 files less than 100 Kb (in Azure Queue) 10 files greater than 100 Kb (in Azure Blob) 20 files in file system (10 files > 100Kb and 10 files <100 Kb)

Test Case Field	Description
Expected Results	1 Uplink test results: 1.1 Files greater than 100 Kb should be moved from Azure Blob to the "ServerOut" folder. 1.2 Files less than 100 Kb should be moved from Azure Queue to the "ServerOut" folder. 1.3 Total 20 files should be created under "ServerOut" folder. 2 Downlink Test results 2.1 Files greater than 100 Kb should be moved to the Azure Blob. (10 files) 2.2 Files less than 100 Kb should be moved to the Azure Queue. (10 files) 2.3 Azure queue count should be 30, which includes original files (10 files) and metadata files belong to original files exist in Azure Blob and Queue (20 files) No error logged in under CPIF error log file Info log file should contains file names and data download and upload statuses from the Azure Blob and queue
Post-Condition	Azure Blob should consist with transferred 10 files Azure service bus queue should consist with 30 messages ServerOut folder should contains total 20 files
Status (Fail/Pass)	Pass
Notes/Comments/Questions:	N/A
Attachments/References	N/A

Table 4.10 Reliability testing test case 6

4.4 Performance Testing

The CPIF middleware solution required to be tested with high, medium and low data loads and different file sizes to verify the performance.

However, it is must to baseline the system requirements which includes all the hardware, software and network bandwidth requirements before starting any performance test, that can be impact the application performance.

Hardware and Software Baseline Requirements

Processor	Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz, 2000 Mhz, 2 Core(s), 4 Logical Processor(s)
Installed Physical Memory (RAM)	8.00 GB
Total Virtual Memory	13.3 GB
System Type	x64-based PC
OS Name	Microsoft Windows 8.1
Network bandwidth (speed)	20 Mbps
Microsoft Azure Region	North Central US

Table 4.11 Performance testing environment settings

Following performance tests are executed to measure the performance statistics of CPIF solution.

Note: Azure Plug-in is utilized to execute the following performance tests.

Performance Test -1	Test the performance of uplink dataflow. File sizes < 100 KB
Performance Test -2	Test the performance of uplink dataflow. File sizes >100 KB
Performance Test -3	Test the performance of downlink dataflow. File sizes < 100 KB
Performance Test -4	Test the performance of downlink dataflow. File sizes >100 KB
Performance Test -5	Test the performance of uplink and downlink dataflows parallelly with mixed file sizes

Table 4.12 Performance test dataset

Performance Test -1

<table>
<tr><td>Test Scenario</td><td>Test the performance of uplink dataflow. File sizes < 100 KB</td></tr>
<tr><td>Description</td><td>CPIF server is stopped before starting this test.
Then the 500 files of less than 100 KB files are placed under the “ServerIn” folder.
Afterwards, CPIF server will be started.
Monitor ServerIn folder, Azure Queue and log files until the test ends</td></tr>
<tr><td>File set</td><td>100 files with each 5KB in size (5 KB X 100)
100 files with each 10KB in size (10KB X 100)
100 files with each 20KB in size (20 KB X 100)
100 files with each 50KB in size (50 KB X 100)
100 files with each 90KB in size (90 KB X 100)</td></tr>
<tr><td>Total size of file set</td><td>~17 MB</td></tr>
<tr><td>Test Results</td><td>- All files are transferred to the Azure queue
- No errors logged in the error log file
- No files remained at ServerIn folder
- 1000 Messages are available in Azure Queue (500 orginal files + 500 Metadata files)
-</td></tr>
<tr><td>Performance Statistics</td><td><table><tr><td>File Transfer Duration</td><td>2.5 Minutes</td></tr><tr><td>File losses</td><td>0</td></tr></table></td></tr>
</table>

Table 4.13 Performance test case 1

Performance Test -2

<table>
<tr><td>Test Scenario</td><td>Test the performance of uplink dataflow. File sizes >100 KB</td></tr>
<tr><td>Description</td><td>CPIF server is stopped before starting this test.
Then the 500 files of greater than 100 KB files are placed under the “ServerIn” folder.
Afterwards, CPIF server will be started.
Monitor ServerIn folder, Azure Blob, Queue and log files until the test ends</td></tr>
<tr><td>File set</td><td>100 files with each 110KB in size (110 KB X 100)
100 files with each 200KB in size (200KB X 100)
100 files with each 500KB in size (500 KB X 100)
100 files with each 1MB in size (1 MB X 100)
100 files with each 2MB in size (2 MB X 100)</td></tr>
<tr><td>Total size of file set</td><td>~380 MB</td></tr>
<tr><td>Test Results</td><td>- All files are transferred to the Azure queue
- No errors logged in the error log file
- No files remained at ServerIn folder
- 500 Files are available in Azure Blob
- 500 Messages are available in Azure Queue (500 Metadata files)
-</td></tr>
<tr><td>Performance Statistics</td><td><table><tr><td>File Transfer Duration</td><td>1 Hour and 7 Minutes</td></tr><tr><td>File losses</td><td>0</td></tr></table></td></tr>
</table>

Table 4.14 Performance test case 2

Performance Test -3

<table>
<tr><td>Test Scenario</td><td>Test the performance of downlink dataflow. File sizes < 100 KB</td></tr>
<tr><td>Description</td><td>CPIF server is stopped before starting this test.
Then the 500 files of less than 100 KB files are already enqueued in the Azure queue and “ServerOut” folder should be empty
Afterwards, CPIF server will be started.
Monitor ServerOut folder, Azure Queue and log files until the test ends</td></tr>
<tr><td>File set</td><td>100 files with each 5KB in size (5 KB X 100)
100 files with each 10KB in size (10KB X 100)
100 files with each 20KB in size (20 KB X 100)
100 files with each 50KB in size (50 KB X 100)
100 files with each 90KB in size (90 KB X 100)</td></tr>
<tr><td>Total size of file set</td><td>~17 MB</td></tr>
<tr><td>Test Results</td><td>- All files are downloaded from Azure queue to the ServerOut folder
- No errors logged in the error log file
- No files remained at Azure queue
- 500 files are available in ServerOut
-</td></tr>
<tr><td>Performance Statistics</td><td><table><tr><td>File Transfer Duration</td><td>1 Minute and 55 seconds</td></tr><tr><td>File losses</td><td>0</td></tr></table></td></tr>
</table>

Table 4.15 Performance test case 3

Performance Test -4

<table>
<tr><td>Test Scenario</td><td>Test the performance of downlink dataflow. File sizes >100 KB</td></tr>
<tr><td>Description</td><td>CPIF server is stopped before starting this test.
Then the 500 files of greater than 100 KB files enqued to Azure Blob (through CPIF solution)
Afterwards, CPIF server will be started.
Monitor ServerOut folder, Azure Blob, Queue and log files until the test ends</td></tr>
<tr><td>File set</td><td>100 files with each 110KB in size (110 KB X 100)
100 files with each 200KB in size (200KB X 100)
100 files with each 500KB in size (500 KB X 100)
100 files with each 1MB in size (1 MB X 100)
100 files with each 2MB in size (2 MB X 100)</td></tr>
<tr><td>Total size of file set</td><td>~380 MB</td></tr>
<tr><td>Test Results</td><td>- All files are downloaded to the ServerOut folder
- No errors logged in the error log file
- No files remained at Azure Blob
- 500 Files are available in ServerOut folder
-</td></tr>
<tr><td>Performance Statistics</td><td><table><tr><td>File Transfer Duration</td><td>55 Minutes</td></tr><tr><td>File losses</td><td>0</td></tr></table></td></tr>
</table>

Table 4.16 Performance test case 4

Performance Test -5

<table>
<tr><td>Test Scenario</td><td>Test the performance of uplink and downlink dataflows parallelly with mixed file sizes</td></tr>
<tr><td>Description</td><td>CPIF server is stopped before starting this test.
Then the 500 files with mixed file sizes enqued to Azure Queue and Blob (through CPIF solution)
500 files with mixed file sizes are placed under “ServerIn” folder.

Afterwards, CPIF server will be started.
Monitor ServerOut folder, Azure Blob, Queue and log files until the test ends</td></tr>
<tr><td>File set</td><td><u>Uplink file load (Already placed under ServerOut folder)</u>
100 files with each 50KB in size (50 KB X 100)
100 files with each 90KB in size (90 KB X 100)
100 files with each 200KB in size (200 KB X 100)
100 files with each 500KB in size (500 KB X 100)
100 files with each 1MB in size (1 MB X 100)

<u>Downlink file load (Already placed under Azure Queue and Blob)</u>
100 files with each 50KB in size (50 KB X 100)
100 files with each 90KB in size (90 KB X 100)
100 files with each 200KB in size (200 KB X 100)
100 files with each 500KB in size (500 KB X 100)
100 files with each 1MB in size (1 MB X 100)</td></tr>
<tr><td>Total size of file set</td><td>Uplink fileload size ~182 MB
Downlink fileload size ~182 MB</td></tr>
<tr><td>Test Results</td><td>- 500 files are downloaded to the ServerOut folder
- 300 files are available in Azure Blob
- 1000 messages are available in Azure Queue (200 original files + 800 meta data files)</td></tr>
<tr><td>Performance Statistics</td><td><table><tr><td>File Transfer Duration for uplink</td><td>52 Minutes</td></tr><tr><td>File Transfer Duration for downlink</td><td>42 minitues</td></tr><tr><td>File losses</td><td>0</td></tr></table></td></tr>
</table>

Table 4.17 Performance test case 5

Chapter 5 Conclusion

5.1 Introduction

This section provides you the summary of how CPIF solution succeeds toward achieving its design objectives. In addition to that, this section also includes existing deficiencies of CPIF solution and the further improvements which could make on top of CPIF solution.

5.2 Summary of Results

This section summarizes how each objective is met through the development of CPIF solution. In the Introduction chapter, it was clearly mentioned all the detailed objective of developing the CPIF solution under the Motivation subsection. Therefore, high-level objective and how there were met describe in below table.

Project Objective	**How CPIF Solution Met Its Objectives**
Client applications are highly coupled with the cloud platforms which are integrated; therefore the flexibility of changing the cloud platform or integrating to another cloud platform is extremely difficult. As the primary objective, the CPIF solution should address the above problem	CPIF solution is developed by accommodating a loosely coupled design by adhering to the Separation of Concern design principle. It utilizes the Plug-in Architecture pattern. Therefore, it provides the flexibility of integrating different cloud platforms without affecting existing components. Additionally, the difficulty on switching between different cloud platforms became extremely easier since it only requires configuration changes to switch to a another cloud platform.
Cost of integrating new cloud platform is high since it requires development and quality assurance effort.	CPIF framework and its respective plug-in components contain the cloud platform integration methodologies. The quality assurance of CPIF solution is already completed. Therefore, if the respective plug-in components for the particular cloud platform is already developed, there will not be any additional cost involves for the integration.

Project Objective	How CPIF Solution Met Its Objectives
Providing Cloud vendor independent framework is another objective of developing the CPIF solution.	CPIF framework exposes generic set of interfaces where any application could directly integrate with the framework rather concerning about the Cloud technology. Therefore, by integrating CPIF framework provides consumer application to integrate and transfer data without thinking of the integration methodologies involves with respective cloud platform
Future extensibility of porting new cloud platform CPIF solution without impacting core CPIF framework and the consumer application is an another objective of implementing CPIF solution	CPIF solution is implemented by utilizing the Plug-in architecture. Therefore, the integration technology of each cloud platform resides in respective plug-in component. Each plug-in component port to the core CPIF framework with minimal configuration changes. Therefore, by thinking about extensibility aspects of CPIF solution, integrating new cloud platform became extremely simpler since that can be done without impacting the changes to the core framework or to the other plug-in components
CPIF solution should support offline capabilities. Which means the CPIF solution should host and executes independently; Therefore, all the data transfer communication can be done in asynchronous mode without maintaining a synchronous connectivity with consumer application. The ultimate objective is bottlenecks of consumer application should not flow into the CPIF middleware solution, vice-versa.	CPIF solution is implemented in a way that can be self-hosted as an executable component or either as a windows service. As per the design integration of CPIF solution is done through windows directory storage. The consumer application is completely disconnected through the CPIF solution since the data communications happen through the configured send and receive directory paths.

Project Objective	How CPIF Solution Met Its Objectives
CPIF solution should be able to integrate with any consumer application which is implemented using any technology (E.g. Java, PHP, etc). Therefore, integration of CPIF solution should be technology independent.	The CPIF solution is a middleware solution which is implemented using Microsoft .NET based technologies. Since it is a middleware solution it is beneficial if it provides the flexibility on integrating consumer applications with different technologies. CPIF solution is designed and implemented in way that integration happened through offline directory storages. Therefore, consumer applications which are implemented using any technology should be able to communicate effectively through the CPIF middleware solution without any issues.

Table 5.1 Results summary

5.3 Deficiencies of CPIF Solution

This section provides the summary of issues exist in CPIF solution.

Issue/limitations	Resolution
Failures in network during data transfer may cause consumer to resend files. Currently, CPIF solution does not contain any retry mechanism. Usually, this issue applies when transferring large files	Consumer application requires monitoring the status of delivering files and if any error occurs, particular file need to be resent.
Currently, CPIF solution is limited to use one cloud platform after the integration.	Hosting two instances of CPIF solution with respective cloud platform plug-ins will solve this issue. Therefore, one instance will take care of data transfer to a one cloud platform while other instance will take care of transferring data to the other cloud platform

Table 5.2 CPIF solution limitations

5.4 Future Improvements of CPIF Solution

This section describes future improvements of CPIF solution.

- Implement more plug-in components which represent different cloud platforms by following CPIF plug-in component design describes in Methodology chapter.
  As of today, there are different cloud vendors in the industry in addition to the Microsoft Azure an Amazon Web Services. Some of them are IBM Cloud, WSO2 Cloud, etc. By developing plug-in components for those Cloud platforms will increase the overall market value of CPIF solution.

- Within the scope of CPIF solution, it covers data transfers to Queue storages and Blob (Block Blob) storages in Azure. (Queue and Bucket in AWS). However, there is more data storage mechanisms exist in Cloud platforms. For an example Azure Topics, Azure Append Blobs, Page Blobs and Azure Table Storages are some of the data storage mechanism. By utilizing existing CPIF design architecture, the CPIF solution could be extended to use the above mentioned additional storage mechanism. That will help to utilize CPIF solution for different purposes of data transfers.

- Currently CPIF middleware solution hosted independently either as Windows executable component or either a Windows Service. Integration happened through the configured directory storages. This will restrict consumer application to hosted in same network environment in order to access the configured directory storages.

  Therefore, as a future extensibility of CPIF solution, Web API could be developed and expose with set of API contracts by wrapping up the existing CPIF framework implementation instead of hosting as Windows executable or Windows Service. This enables CPIF solution to host in different server environment with different network rather hosting on same network where consumer application hosted. Therefore, consumer applications directly connect with Web API and the Web API is responsible of sending and receiving data streams with Base64 Encryption mechanism and communicates with the CPIF framework.

Appendix

5.5 Azure Storage queues and Service Bus queues

When developing CPIF Azure plug-in components, there are two types of queues available. However, as evaluated only one type of queue is selected which is Service Bus queue. Following table represents the detailed comparison of Azure Storage queue and Service Bus queue.

Comparison Criteria	**Storage queues**	**Service Bus queues**
Ordering guarantee	No	Yes - First-In-First-Out (FIFO)
Delivery guarantee	At-Least-Once	At-Least-Once At-Most-Once
Atomic operation support	No	Yes
Batched receive	Yes	Yes
Batched send	No	Yes

Table 0.1 Azure Service Bus queue vs. Storage queue

In addition to the above comparison, usually the Storage Queues are utilized when an application requires storing over 80 GB of messages in a queue. Due to large set of messages application could track the progress for processing a message inside of the storage queue. Also it may require the server side logs for all of the transactions which are getting executed against the storage queues.

In contrast, Service Bus queues able to receive messages without polling the queue and it support TCP-based protocol. Therefore, we can implement transactional behavior to achieve atomicity when communicating with multiple messages with service bus queue. Additional advantage is it provides guaranteed first-in-first-out (FIFO) ordered delivery. However, the one limitation is service bus queue size should not grow larger than 80 GB. Due to these advantages Service Bus queue is selected to implement the Azure Plug-in component by assuming client applications does not have a requirement of sending over 80GB of messages to a single queue.

5.6 Azure Block Blob vs Page Blob vs Append Blob

Azure storage has following three types of blobs.

- Block blob
- Page blob
- Append blob

The following table provides a comparison between them.

Block Blob	**Page Blob**	**Append Blob**
Block Blobs are comprised of blocks and each Block is identifiable by a Block ID	Page blobs are collection of pages that are optimized for random read write operations.	Append blobs are similar to block blobs but are optimized for Append operations
Usually used for streaming Sequential Data like Video	Usually used for non-Sequential Read and Write.	Usually used for activities like Logging
Each Block can be up to 4 MB	Page can be up to 512 bytes	Each Block can be up to 4 MB
Up to 50,000 Blocks can be created.	No limitations on the number of Pages created	Up to 50,000 Blocks can be created.
Blocks can be uploaded in any order and need to commit the blocks by sending the order at the end of the process. (A.K.A two-step block upload-then-commit process).	Any writes that are done get committed immediately (in-place process)	Cannot Update or delete the existing blocks in a blob.
Any uncommitted blocks will be deleted after a week time period or another blob with same name is created with commit process	-NA-	-NA-
Any Uncommitted block can be over written by using the same block ID.	Write operation can overwrite a page or a number of pages.	Updating or overwriting a block is not possible.
Maximum Size of block blob : 195 GB Maximum uncommitted blobs :100000 (Max size :20000MB)	Maximum size of page blob : 1TB	Maximum size of page blob : 195 GB

Multiple clients writing to same blob is not possible (synchronization needed)	Multiple clients writing to same blob is not possible (synchronization needed)	Multiple clients writing to same blob is possible (no synchronization needed)

Table 0.2 Azure blob comparison

Reference:

[1] Vijaya Manikandan. (2015). Block Blob vs Page blob vs Append Blob. Retrieved April 4, 2019 from http://www.techxperiments.com/2015/10/14/block-blob-vs-page-blob-vs-append-blob/

Reference [1] verified through following references [2] and [3].

[2] Microsoft Azure. (2019) Understanding Block Blobs, Append Blobs, and Page Blobs. Retrieved April 4, 2019 from https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs

[3] Microsoft Azure. (2010) Using Windows Azure Page Blobs and How to Efficiently Upload and Download Page Blobs. Retrieved April 4, 2019 from https://blogs.msdn.microsoft.com/windowsazurestorage/2010/04/10/using-windows-azure-page-blobs-and-how-to-efficiently-upload-and-download-page-blobs/

By evaluating above comparison, Block Blob storage is selected for the CPIF Azure Plug-in development since our requirement is only to access to the files, but not to change them inside the blob storage. Also Block Blobs more suitable for storing user specific files compare to other Blob types.

References

[1]. Prof Dr, Claudia Müller-birn. (2012). Cloud Computing. Retrieved April 4, 2019 from http://citeseerx.ist.psu.edu/viewdoc/citations?doi=10.1.1.462.4311

[2]. University of California at Berkeley. (2009). Above the Clouds. Retrieved January 14, 2019 from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[3]. Saurabh Gupta, Saurabh Gupta, Saurabh Gupta. (2012). Microsoft Azure vs Amazon EC2. Retrieved January 4, 2019 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.471.7343&rank=1

[4]. Microsoft Azure. (2019). Azure Storage scalability and performance targets for storage accounts. Retrieved April 4, 2019 from https://docs.microsoft.com/en-us/azure/storage/common/storage-scalability-targets

[5]. Microsoft Azure. Azure Storage Documentation. Retrieved January 4. 2019 from https://docs.microsoft.com/en-us/azure/storage/

[6]. Microsoft Azure. Azure Service Bus Messaging Documentation. Retrieved April 4, 2019 from https://docs.microsoft.com/en-us/azure/service-bus-messaging/

[7]. Microsoft Azure. Hybrid Cloud Architectures with AWS. Retrieved April 4, 2019 from https://aws.amazon.com/enterprise/hybrid/

[8]. Microsoft Azure. (2017). Managed Extensibility Framework (MEF). Retrieved February 2, 2019 from https://docs.microsoft.com/en-us/dotnet/framework/mef/

[9]. Microsoft Azure. (2019). Azure Storage scalability and performance targets for storage accounts. Retrieved April 6, 2019 from https://docs.microsoft.com/en-us/azure/storage/common/storage-scalability-targets

[10]. Microsoft Azure. (2019). Storage queues and Service Bus queues - compared and contrasted. Retrieved April 6, 2019 from https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted

[11]. Microsoft Azure. (2019) Understanding Block Blobs, Append Blobs, and Page Blobs. Retrieved April 4, 2019 from https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs

[12]. Microsoft Azure. (2010) Using Windows Azure Page Blobs and How to Efficiently Upload and Download Page Blobs. Retrieved April 4, 2019 from https://blogs.msdn.microsoft.com/windowsazurestorage/2010/04/10/using-windows-azure-page-blobs-and-how-to-efficiently-upload-and-download-page-blobs/

[13]. Amazon AWS. (2019). Amazon Simple Storage Service API Reference . Retrieved April 30, 2019 from https://docs.aws.amazon.com/AmazonS3/latest/API/s3-api.pdf

[14]. Amazon AWS. (2006). Amazon Simple Storage Service Developer Guide. Retrieved April 30, 2019 from https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-dg.pdf

[15]. Amazon AWS. (2012). Amazon Simple Queue Service API Reference . Retrieved April 30, 2019 from https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/sqs-api.pdf

[16]. Amazon AWS. (2019). Amazon Simple Queue Service Developer Guide. Retrieved April 30, 2019 from https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dg.pdf