z/OS Connect Enterprise Edition V3.0

Performance Summary using the API Requester

Version 1.0

June 2018

Alan Hollingshead
alan_hollingshead@uk.ibm.com

z/OS Connect EE
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

1.1 Notices

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of z/OS Connect EE V3.0. The information is not intended as the specification of any programming interfaces that are provided by z/OS Connect EE.

It is assumed that the reader is familiar with the concepts and operation of z/OS Connect EE V3.0.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed “asis”. The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the customer to evaluate this data and project the results to their operational environment.

The performance data contained in this report was measured in a controlled environment and results obtained in other environments may vary significantly.

1.2 Acknowledgements

Credit to the IBM China team, in particular Yan Feng, for her help and support.

1.3 Trademarks and service marks

1.4 Terminology

CICS	-	CICS Transaction Server
CICS PA	-	CICS Performance Analyzer
Cost per transaction (ms)	-	CPU usage per transaction, in milliseconds
CP	-	Central Processor
CPU %	-	Percentage of CPU time used by transactions running on general purpose processors
EE	-	Enterprise Edition
GCP	-	General purpose Central Processor
IMS	-	IMS region
JIT optimization	-	Just-In-Time optimization. The JIT compiles the JVM bytecode to machine code at runtime and uses optimization techniques such as moving calculations and caching them, moving methods to be inline, removing unnecessary locks, thus allowing the code to become smaller and faster.
PID	-	Product Identification Number
RMF	-	Resource Measurement Facility
SMF	-	System Management Facility
SSL	-	Secure Sockets Layer
Swagger 2.0	-	Open source software framework used to describe the structure of APIs.
TPS	-	Number of Transactions Per Second
TT	-	Think Time in seconds. The time between individual requests.
Workload Driver	-	An application written for the purpose of these performance tests to simulate multiple simultaneous client requests.
zIIP	-	IBM z Systems Integrated Information Processor
z/OS application		For the purposes of this report, the z/OS application may be a CICS, IMS or batch application.

2. Overview

This report contains performance measurements for z/OS Connect EE V3.0, program number (PID) 5655-CEE, using the API requester feature of z/OS Connect EE V3.0.

What is z/OS Connect EE?

Cloud and mobile applications reshape the way enterprises and systems interact. RESTful APIs that use JSON message formats are the predominant standards for new application development. IBM® z/OS® Connect Enterprise Edition provides a framework that enables z/OS-based programs and data to participate fully in the new API economy for mobile and cloud applications.

IBM z/OS Connect Enterprise Edition V3.0 provides RESTful API access to z/OS applications and data hosted in subsystems such as CICS®, IMS™, and Db2. The framework provides concurrent access, through a common interface, to multiple z/OS subsystems.

IBM z/OS Connect Enterprise Edition V3.0 also provides the capability that allows z/OS-based programs to access any RESTful endpoint, inside or outside z/OS, through RESTful APIs with JSON formatted messages. The framework supports applications on CICS, IMS, and z/OS programs to call RESTful APIs through z/OS Connect EE. This capability is provided through the API requester feature of z/OS Connect EE and is the focus of this report.

What is the API requester feature in z/OS Connect EE?

With z/OS Connect EE, z/OS applications can consume RESTful APIs, thus utilizing the underlying services and data from external hosts (also called request endpoints). In this scenario, z/OS acts as a RESTful API consumer.

An intermediary named API requester is required to facilitate RESTful API calls by z/OS-based programs.

Figure 1 shows how z/OS Connect EE connects z/OS applications with services and data available from a request endpoint.

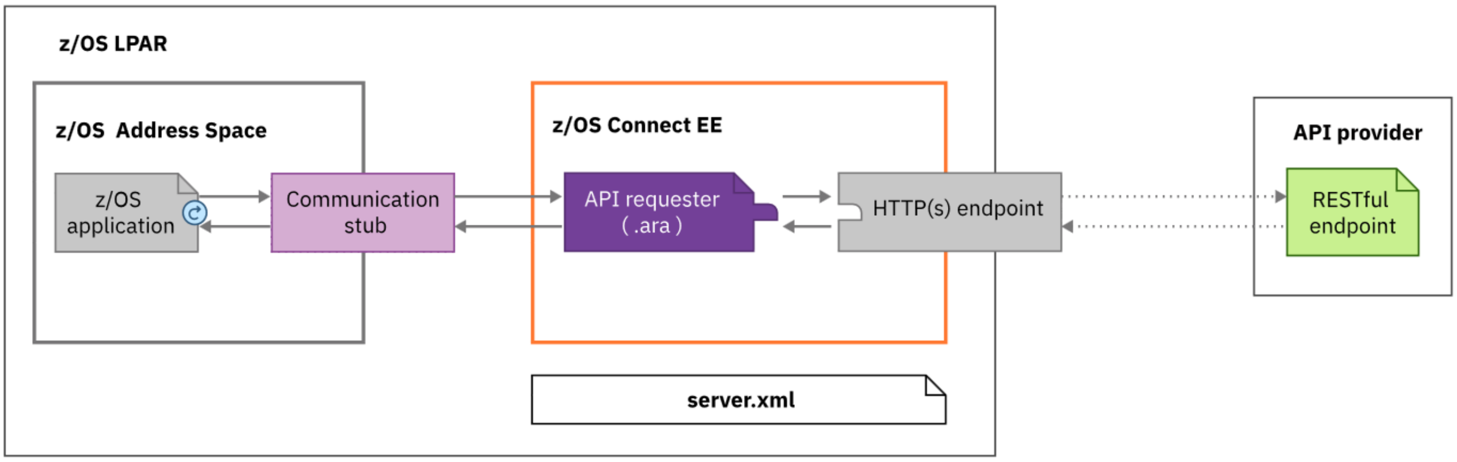


Figure 1 API requester flow

3. Test Environment

This report focuses on an end-to-end solution that uses a client z/OS application running in a z/OS address space (for example, CICS, IMS, or simply a native z/OS application) to make requests to an API provider through a z/OS Connect EE server.

Although a CICS API requester application was used in this test environment, the measurements taken and results shown in section “9 Results” equally apply to IMS and other z/OS applications using the API requester capability.

The focus of this report is to provide a baseline. It does not cover any of the security options available with the API requester which would naturally demand additional system resources beyond this baseline.

Figure 2 shows the flow of requests from the workload driver and responses from the API Provider for the scenario:

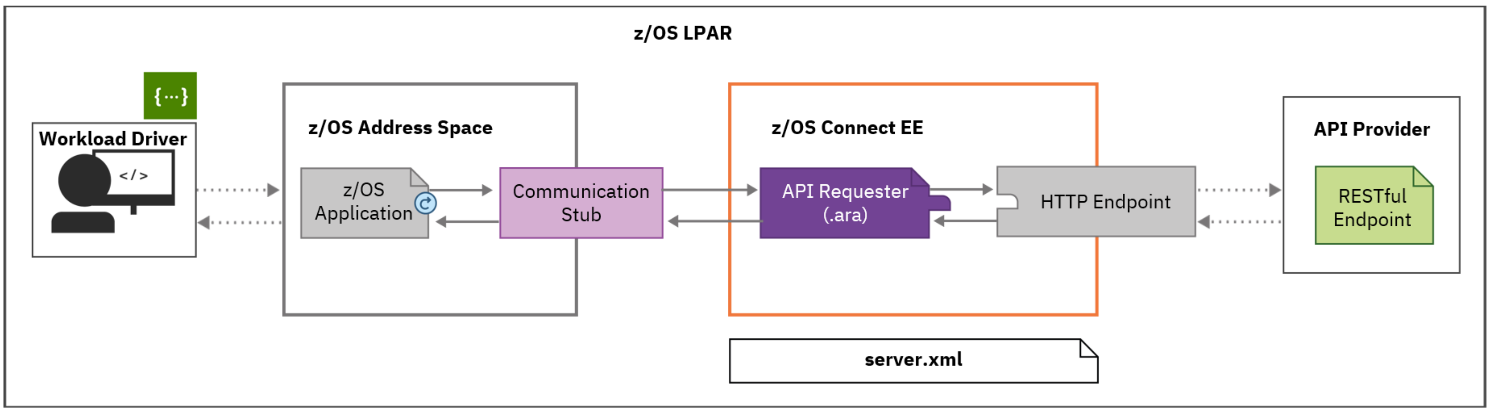


Figure 2 End-to-end solution using the API requester

3.1 Flow of work

The flow of work is as follows:

1. The workload driver sends concurrent client requests to the client z/OS application.
2. The client z/OS application contains data structures for the request, response and API information file, and populates the request structure to be passed to the API provider.
3. The client z/OS application makes a call using the Communication Stub to z/OS Connect EE.
4. The API requester feature of z/OS Connect EE converts the incoming request from the Communication Stub into an API request using JSON format.
5. z/OS Connect EE makes a call to remote API provider using the JSON request via HTTP.
6. The API provider processes the request and responds to z/OS Connect EE.
7. z/OS Connect EE processes the response, formats it, and sends it back to the Communication Stub and the client z/OS application.

3.2 Hardware

- IBM System z13 2964-NE1 model 7A5
- 10GB of Central Storage (RAM)
- LPAR with 6 dedicated GCPs (no zIIPs)
- OSA-Express5S 10GB Ethernet

3.3 Software

- z/OS V2.1
- z/OS Connect Enterprise Edition (EE) V3.0.8
  - z/OS Connect EE build 20180416-1517
  - Liberty 18.0.0.1 with Angel Process V7
- IBM 64-bit SDK for z/OS Java Technology Edition, Version 8.0
  - Java 1.8 SR5 FP7
- CICS Transaction Server V5.4

3.4 Workload Driver

The workload driver used to simulate multiple client requests is a Java application that runs in its own JVM on the same LPAR as the other z/OS address spaces. This is to minimise any network latency that could affect performance results.

Thinktime

Requests are issued by the workload driver at a regular pace, using a “thinktime”. For each client this is the time the workload driver waits after a response has been received before sending the next request. All performance runs in this report use a thinktime of 200ms to simulate a typical customer environment.

HTTP POST Requests

All scenarios issue HTTP POST requests to call the API provider.

3.5 Asymmetric Payloads

Each payload was asymmetric, meaning that the size of data for the request was different to the size of data for the response. For example, a 16 byte payload request may have generated an 8K response. The size of the response was dependent upon the data within the request.

3.6 Performance measurement procedure

The same procedure was carried out for each measurement and consisted of the following steps:

1. Liberty server hosting the API was started.
2. z/OS Connect EE server was started with “clean” option.
3. client z/OS application started.

4. Workload driver started.
5. The workload to be measured rapidly reached a steady throughput for requests (transactions) per second. To ensure JIT optimisation, several hundred thousand requests were run before any measurements were taken.
6. IBM z/OS Resource Measurement Facility (RMF) data was collected at one minute intervals.

3.7 Performance metrics

The following metrics were gathered during the performance runs for the z/OS Connect EE server:

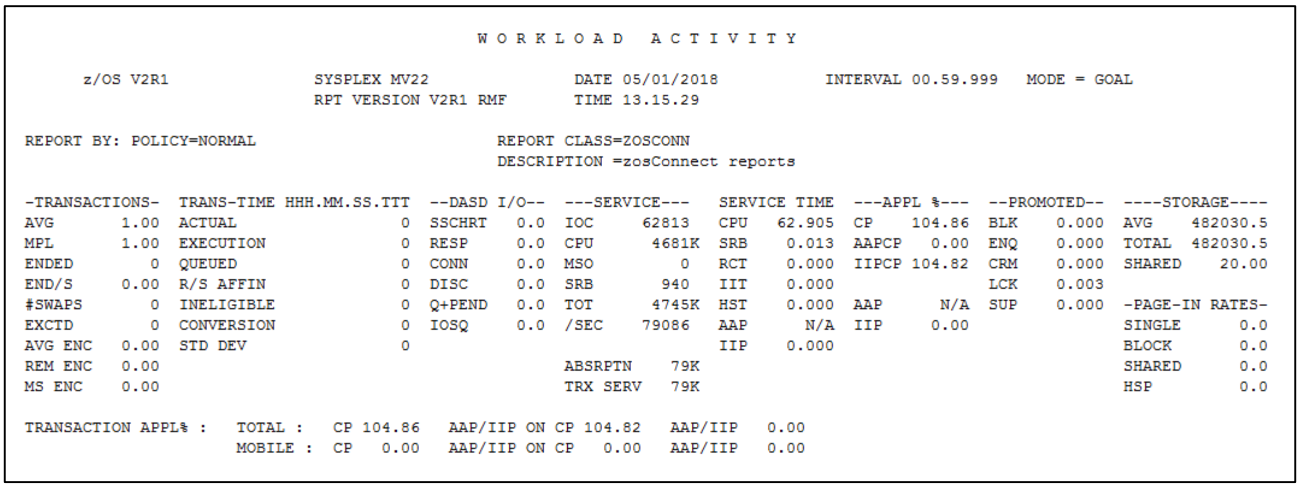
- Request rate (transactions per second)
- Average response time
- CP (general processor) usage
- Potential zIIP usage
- CPU cost per transaction

The request rate (transactions per second) and average response time values were gathered by the workload driver.

CP and potential zIIP usage values were extracted from RMF workload activity data. RMF reports CPU utilisation that is based on uniprocessor capacity, which means that a value of 100% represents the capacity of one processor.

The CPU cost per transaction was calculated by dividing the CPU service time by the number of seconds in the SMF interval. This is then divided by the number of transactions per second (TPS) for the same SMF interval, and finally multiplied by one thousand to get the result in milliseconds.

Figure 3 shows an example of a Workload Activity extract from an RMF report for a z/OS Connect EE server. For the one minute interval, the APPL% section shows 104.86% of general processor time was used (the “CP” value) and that 104.82% (the “IIPCP” value) is potentially offloadable to zIIP processors.



```
                                           W O R K L O A D   A C T I V I T Y

      z/OS V2R1              SYSPLEX MV22            DATE 05/01/2018          INTERVAL 00.59.999   MODE = GOAL
                             RPT VERSION V2R1 RMF    TIME 13.15.29

REPORT BY: POLICY=NORMAL                        REPORT CLASS=ZOSCONN
                                                DESCRIPTION =zosConnect reports

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       1.00  ACTUAL                 0  SSCHRT   0.0  IOC     62813   CPU   62.905  CP    104.86  BLK    0.000  AVG    482030.5
MPL       1.00  EXECUTION              0  RESP     0.0  CPU      4681K  SRB    0.013  AAPCP   0.00  ENQ    0.000  TOTAL  482030.5
ENDED        0  QUEUED                 0  CONN     0.0  MSO         0   RCT    0.000  IIPCP 104.82  CRM    0.000  SHARED    20.00
END/S     0.00  R/S AFFIN              0  DISC     0.0  SRB       940   IIT    0.000                LCK    0.003
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.0  TOT      4745K  HST    0.000  AAP      N/A  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC    79086   AAP      N/A  IIP     0.00                SINGLE      0.0
AVG ENC   0.00  STD DEV                0                                IIP    0.000                              BLOCK       0.0
REM ENC   0.00                                          ABSRPTN    79K                                            SHARED      0.0
MS ENC    0.00                                          TRX SERV   79K                                            HSP         0.0

TRANSACTION APPL% :   TOTAL :   CP 104.86   AAP/IIP ON CP 104.82   AAP/IIP   0.00
                      MOBILE :  CP   0.00   AAP/IIP ON CP   0.00   AAP/IIP   0.00
```

Figure 3 Example of RMF Workload Activity extract

4. API

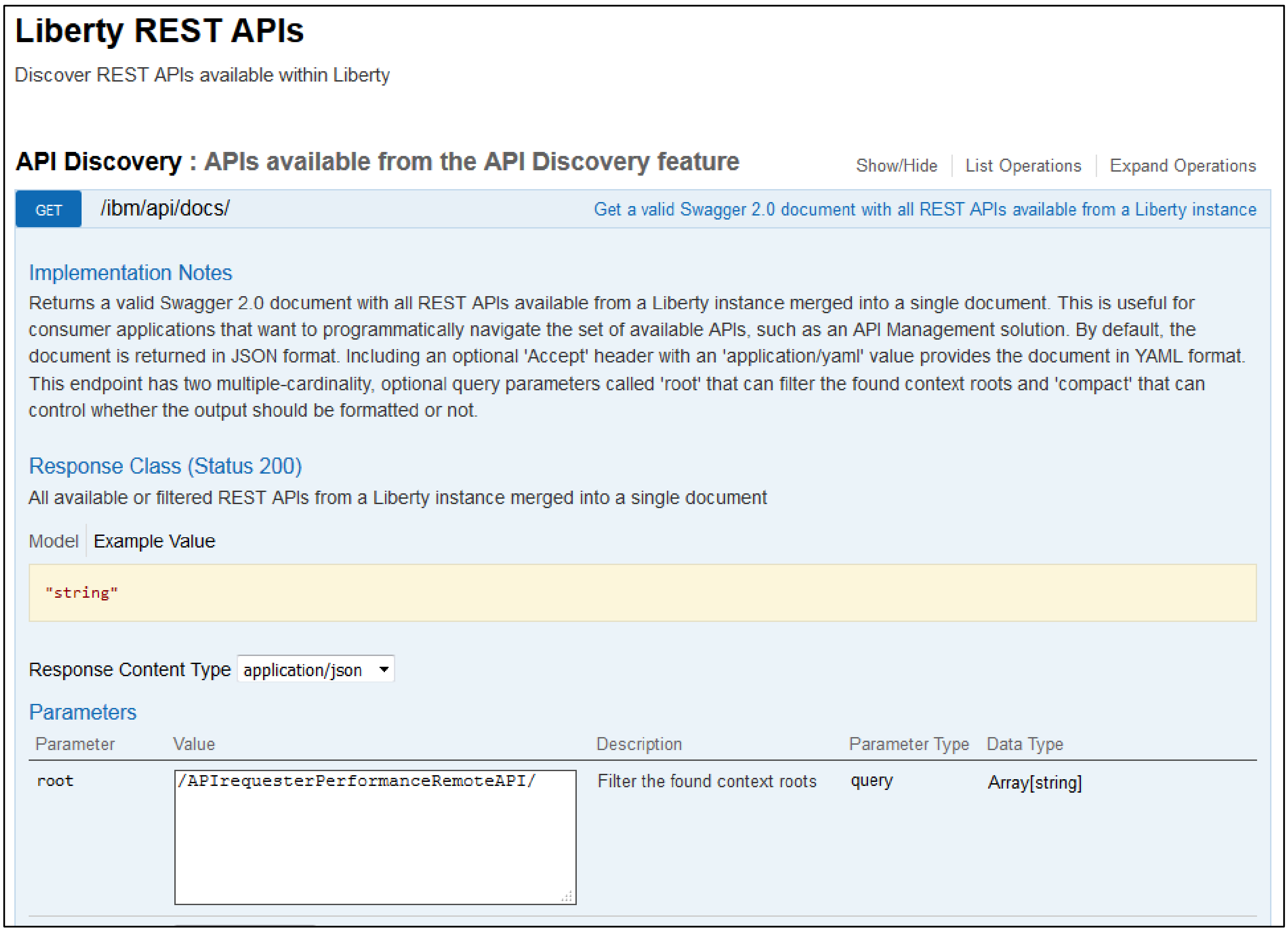
The API provider used an API called **APIrequesterPerformanceRemoteAPI** for the performance runs.

This API was a Liberty API deployed using JAX-RS technology. The API received input from the API requester indicating the size of the payload the API was to respond with. For the purposes of this report, these response payloads were 1K, 4K, or 8K.

4.1 API Swagger file

IBM's API Discovery was used to discover the REST API documentation on the Liberty server for the REST API, **APIrequesterPerformanceRemoteAPI.**

The extracted Swagger 2.0 API in JSON format was called **APIRPERF.json**.



Liberty REST APIs

Discover REST APIs available within Liberty

API Discovery : APIs available from the API Discovery feature

Show/Hide | List Operations | Expand Operations

GET /ibm/api/docs/ Get a valid Swagger 2.0 document with all REST APIs available from a Liberty instance

Implementation Notes

Returns a valid Swagger 2.0 document with all REST APIs available from a Liberty instance merged into a single document. This is useful for consumer applications that want to programmatically navigate the set of available APIs, such as an API Management solution. By default, the document is returned in JSON format. Including an optional 'Accept' header with an 'application/yaml' value provides the document in YAML format. This endpoint has two multiple-cardinality, optional query parameters called 'root' that can filter the found context roots and 'compact' that can control whether the output should be formatted or not.

Response Class (Status 200)

All available or filtered REST APIs from a Liberty instance merged into a single document

Model | Example Value

```
"string"
```

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
root	/APIrequesterPerformanceRemoteAPI/	Filter the found context roots	query	Array[string]

Figure 4 API Discovery used to generate Swagger 2.0 document in JSON format

5. z/OS Connect EE Environment

This chapter describes the z/OS Connect EE environment the performance tests were run in.

5.1 JCL Configuration for z/OS Connect EE

The started procedure for z/OS Connect EE was configured with:

- REGION size 0M
- MEMLIMIT 4G

5.2 JVM Configuration

The JVM used by z/OS Connect EE was configured with:

- IBM 64-bit SDK for z/OS Java Technology Edition, Version 8.0
  - Java 1.8 SR5 FP7
- Maximum Java heap size (Xmx) 1G
- All other JVM system properties use default values (including garbage collection mode: gencon)

5.3 The server.xml Configuration File

The server.xml file used for the z/OS Connect EE server contained the following definitions:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="z/OS Connect EE 3.0. Polling not enabled.">

    <config updateTrigger="mbean"></config>

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:apiRequester-1.0</feature>
        <feature>zosconnect:zosConnectCommands-1.0</feature>
    </featureManager>

    <!—Server port -->
    <httpEndpoint id="defaultHttpEndpoint"
          enabled="true"
          host="127.0.0.1"
          httpPort="2222" />

    <!-- default security setting for all zosConnect managed resources -->
    <zosconnect_zosConnectManager requireAuth="false" requireSecure="false"/>

    <!-- Connection to remote API. The id matches the API requester connectionRef -->
    <zosconnect_endpointConnection id="Server2Ref" host="localhost" port="3188"/>

</server>
```

Figure 5 The server.xml configuration for the z/OS Connect EE server

Feature Definitions

The feature definitions specified in the server.xml configuration file shown in Figure 5 contained two features:

- The first feature, `zosconnect:apiRequester-1.0`, was for the API requester. This automatically installs the z/OS Connect EE V3.0 feature during the initialization of the z/OS Connect EE server.
- The second feature, `zosconnect:zosConnectCommands-1.0`, was to allow SDSF MODIFY commands to enable any changes made to the server.xml configuration file. Note that polling was not enabled in the server.xml configuration file as this can have an impact on performance and is not recommended in a production environment.

zosConnectManager Definition

The `zosconnect_zosConnectManager` definition specified in the server.xml configuration file shown in Figure 5 contained:

- Security elements `requireSecure` and `requireAuth` were disabled as the performance runs did not use any security characteristics.

zosconnect_endpointConnection Definition

The `zosconnect_endpointConnection` definition specified in the server.xml configuration file shown in Figure 5 on page 13 contained:

- `id` value of `Server2Ref`.
- `host` and `port` for the remote API hosted by a Liberty server. For these performance runs, the Liberty server was on the same LPAR as z/OS Connect EE server (to avoid network latency affecting the performance).

API requester

The API requester used for the performance runs was automatically installed during start up of the server, and does not need to be explicitly defined in the server.xml file.

6. z/OS Application

The z/OS application used could have been a CICS, IMS or z/OS batch application.

In the test environment used for these performance runs, we chose a CICS application. For the CICS application to communicate with the API requester, a z/OS Connect EE communication stub is required.

The z/OS Connect EE communication stub is a module that establishes HTTP connections with the z/OS Connect EE server, transfers data between z/OS applications and z/OS Connect EE, and handles status and return codes that are issued by z/OS Connect EE.

Although the CICS communication stub was used in this test environment, the measurements taken and results shown in section “9 Results” equally apply to the non-CICS communication stub.

The z/OS Connect EE CICS communication stub was installed in the CICS region by adding <hlq>.SBAQLIB1 to the DFHRPL and the following resources defined:

- BAQCSTUB PROGRAM – default settings
- BAQQ TDQUEUE – default settings
- BAQURIMP URIMAP – host 127.0.0.1, port 9080, scheme http.

7. API requester artifacts

The z/OS Connect EE build toolkit (zconbt) is supplied as part of the product install as zconbt.zip. After unpacking the zconbt.zip file on zFS, the shell script “zconbt.zos” (specifically for use on z/OS) was used to generate all the artifacts to enable the client z/OS application to call the API (**APIrequesterPerformanceRemoteAPI)** via z/OS Connect EE.

Using a Swagger file and properties file, the build toolkit generated several artifacts including

- an API requester archive file (APIPERF.ara)
- Request and response data structures (API client code)
- API information file.

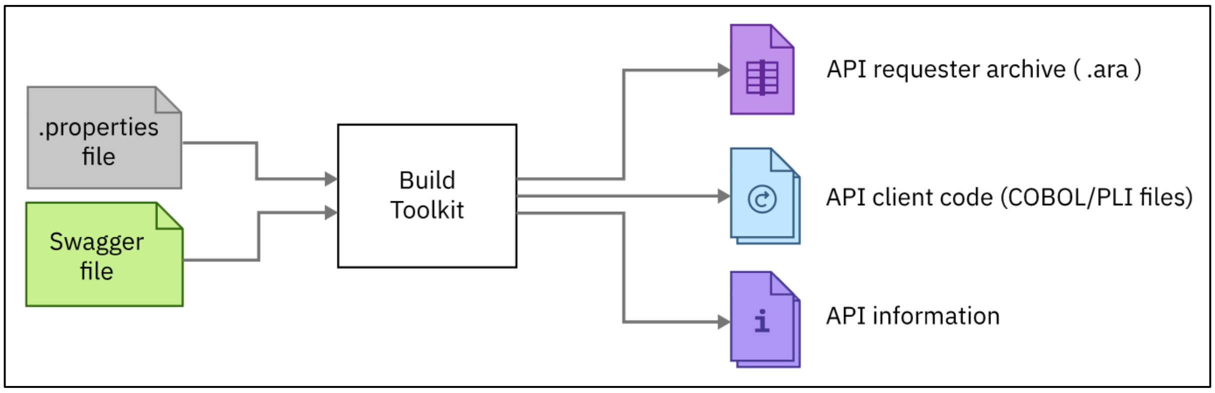

.properties file
Swagger file
Build Toolkit
API requester archive (.ara)
API client code (COBOL/PLI files)
API information

Figure 6 Generating API requester artifacts using the build toolkit

The build toolkit used the Swagger file **APIRPERF.json** obtained in “ 4.1 API Swagger file on page 11”.

A properties file was created, called **APIRPERF.properties**, and specified the following values:

```
apiDescriptionFile=/u/zosconn/btFiles/APIRPERF.json
language=COBOL
dataStructuresLocation=/u/zosconn/btFiles/APIRPerformance
apiInfoFileLocation=/u/zosconn/btFiles/APIRPerformance
requesterPrefix=PER
connectionRef=Server2Ref
logFileDirectory=/u/zosconn/btFiles/APIRPerformance
```

Figure 7 Properties file used by build toolkit command

The properties used were:

- apiDescriptionFile - the name of the JSON Swagger file, APIRPERF.json.
- Language structures were to be generated into COBOL.

- dataStructuresLocation - output directory where the generated request and response data structures, PER00Q01 and PER00P01 respectively (see *Data Structures Generated*), were to be stored.
- apiInforFileLocation - output directory where the generated API information file, PER00I01, was to be stored (see *API Information File Generated*).
- requesterPrefix - PER was used to prefix the generated data structures and API information file.
- connectionRef - set to Server2Ref to identify the RESTful endpoint defined in server.xml (see *zosconnect_endpointConnection Definition* on page 14).
- logFileDirectory - output directory for the build toolkit log file generated.

The build toolkit command run was:

```
./bin/zconbt.zos -p ../APIPERF.properties -f ../APIPERF.ara
```

Figure 8 Build toolkit command to generate API requester artifacts

Data Structures Generated

Figure 9 shows an extract of the request data structure, PER00Q01, generated:

```
       * This file contains the generated language structure(s) for
       *  request JSON schema 'VaryPayload_request.json'.
       * This structure was generated using 'DFHJS2LS' at mapping level
       *  '4.1'.

              06 ReqBody.
                09 count-in                        PIC S9(9) COMP-5 SYNC.
                09 count-out                       PIC S9(9) COMP-5 SYNC.
```

Figure 9 Request data structure, PER00Q01

Figure 10 shows an extract of the response data structure, PER00P01, generated:

```
       * This file contains the generated language structure(s) for
       *  response JSON schema 'VaryPayload_200_response.json'.
       * This structure was generated using 'DFHJS2LS' at mapping level
       *  '4.1'.

              06 RespBody.
                09 recv-size                       PIC S9(9) COMP-5 SYNC.
                09 send-size                       PIC S9(9) COMP-5 SYNC.
                09 taskid                          PIC S9(9) COMP-5 SYNC.
                09 tranid                          PIC X(4).
                09 rs-spare                        PIC X(16).
                09 user-data-num                   PIC S9(9) COMP-5 SYNC.
                09 user-data OCCURS 512            PIC X(32).
```

Figure 10 Response data structure, PER00P01

API Information File Generated

Figure 11 shows an extract of the API information file, PER00I01, generated:

```
          03 BAQ-APINAME                 PIC X(255)
             VALUE 'zCEE-API-requester-Performance-Remote-API_1.0.0'.
          03 BAQ-APINAME-LEN             PIC S9(9) COMP-5 SYNC
             VALUE 47.
          03 BAQ-APIPATH                 PIC X(255)
             VALUE '/APIrequesterPerformanceRemoteAPI/apirRemote/varyRe
      -    'spPayload'.
          03 BAQ-APIPATH-LEN             PIC S9(9) COMP-5 SYNC
             VALUE 60.
          03 BAQ-APIMETHOD               PIC X(255)
             VALUE 'POST'.
          03 BAQ-APIMETHOD-LEN           PIC S9(9) COMP-5 SYNC
             VALUE 4.
```

Figure 11 Extract from API information file, PER00I01

z/OS application

For the client z/OS application to call the API, the application was modified to include the generated request and response data structures, the API information file, and the communication stub to communicate with the z/OS Connect EE server.

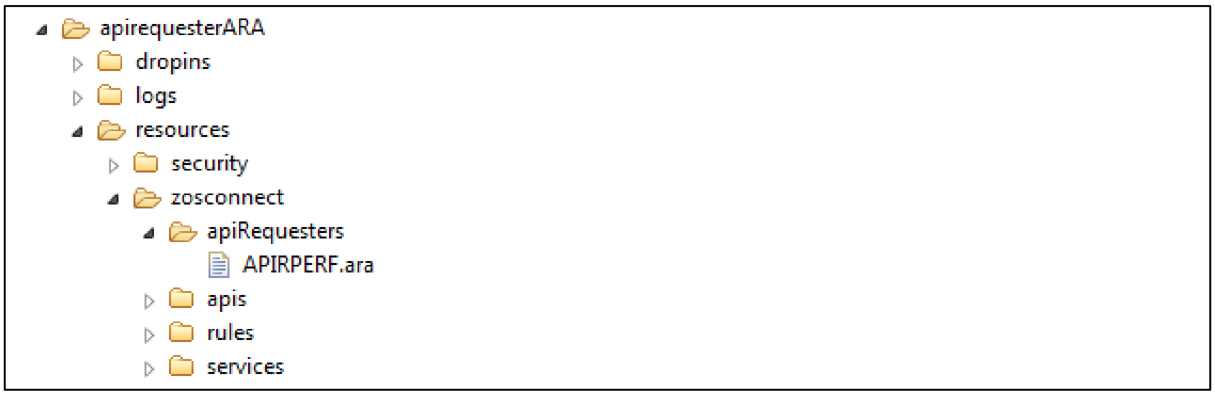
Figure 12 shows an extract of the COBOL client z/OS application used:

```
 : : :

01 REQUEST.
      COPY PER00Q01.
01 RESPONSE.
      COPY PER00P01.
01 API-INFO-OPER1.
      COPY PER00I01.
01 BAQ-REQUEST-INFO.
        : : :
01 BAQ-RESPONSE-INFO.
        : : :

77 COMM-STUB-PGM-NAME          PIC X(8) VALUE 'BAQCSTUB'.

: : :

* Populate value for request to allow different payload response sizes

: : :

* Use pointer and length to specify the location of
*  request and response segment.
     SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.
     MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.
     SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.
     MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.

CALL COMM-STUB-PGM-NAME USING
      BY REFERENCE API-INFO-OPER1
      BY REFERENCE BAQ-REQUEST-INFO
      BY REFERENCE BAQ-REQUEST-PTR
      BY REFERENCE BAQ-REQUEST-LEN
      BY REFERENCE BAQ-RESPONSE-INFO
      BY REFERENCE BAQ-RESPONSE-PTR
      BY REFERENCE BAQ-RESPONSE-LEN.

: : :
```

Figure 12 Extract from client z/OS application

Deploy API requester

The API requester, APIPERF.ara, generated by the build toolkit, was copied to the *apiRequesters* directory for the z/OS Connect EE server:



```
apirequesterARA
    dropins
    logs
    resources
        security
        zosconnect
            apiRequesters
                APIRPERF.ara
            apis
            rules
            services
```

Figure 13 The location of the APIPERF.ara file in the z/OS Connect EE server directory structure

The API requester, APIPERF.ara, was automatically installed during the initialization of the z/OS Connect EE server, see *Figure 14*.

```
BAQR1102I: z/OS Connect EE API requester archive APIRPERF.ara is installed and
API requester zCEE-API-requester-Performance-Remote-API_1.0.0 is deployed suc-
cessfully.
```

Figure 14 Extract from z/OS Connect EE messages log during server initialization

8. Format of Requests and Responses

The format of the 16 byte request (referenced by `BAQ-REQUEST-PTR` of length `BAQ-REQUEST-LEN`) was:

- X'000000010000002' for the API to return a 1K response
- X'000000010000008' for the API to return a 4K response
- X'000000010000010' for the API to return an 8K response.

These mapped to the two fields, count_in and count_out, of the request data structure, PER00Q01, as shown in Figure 9 on page 17.

The format of the response (referenced by `BAQ-REQUEST-PTR` of length `BAQ-REQUEST-LEN`) for the 1K response was:

```
..................KCSMI...................0001-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0002-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0003-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0004-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0005-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0006-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0007-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0008-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0009-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0010-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0011-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0012-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0013-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0014-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0015-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0016-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0017-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0018-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0019-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0020-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0021-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0022-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0023-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0024-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0025-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0026-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0027-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0028-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0029-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-0030-ABCDEFGHIJKLMNOPQRSTUVWXYZ-0031-ABCDEFGHIJKLMNO

PQRSTUVWXYZ-
```

Figure 15 Format of 1K response

The format of the 4K and 8K responses followed a similar pattern. For 4K responses there were 128 fields of the form "`00nn-ABCDEFGHIJKLMNOPQRSTUVWXYZ`", whilst for 8K responses there were 256 fields in this format.

9. Results

The following sections of this report show the results of the performance runs.

Disclaimer: All performance data this is contained in this report was obtained in the specified operating environment and configurations, and must be considered as an example. Performance characteristics of other operating environments might differ.

IBM does not represent, warrant, or guarantee that the same or similar results will be achieved in a user's environment as the results that are shown in this report.

9.1 Transactions Per Second

The performance runs simulated:

- 100 clients
- 150 clients
- 200 clients
- 250 clients
- 300 clients

The performance runs used the following payload sizes for each response in turn:

- 1K response
- 4K response
- 8K response

Although z/OS Connect EE supports payload sizes much larger than 8K, the CPU resources available for the LPAR used for these performance runs became constrained when larger payloads with large numbers of clients were used. This is due to available CPU capacity and is not a limitation of the product.

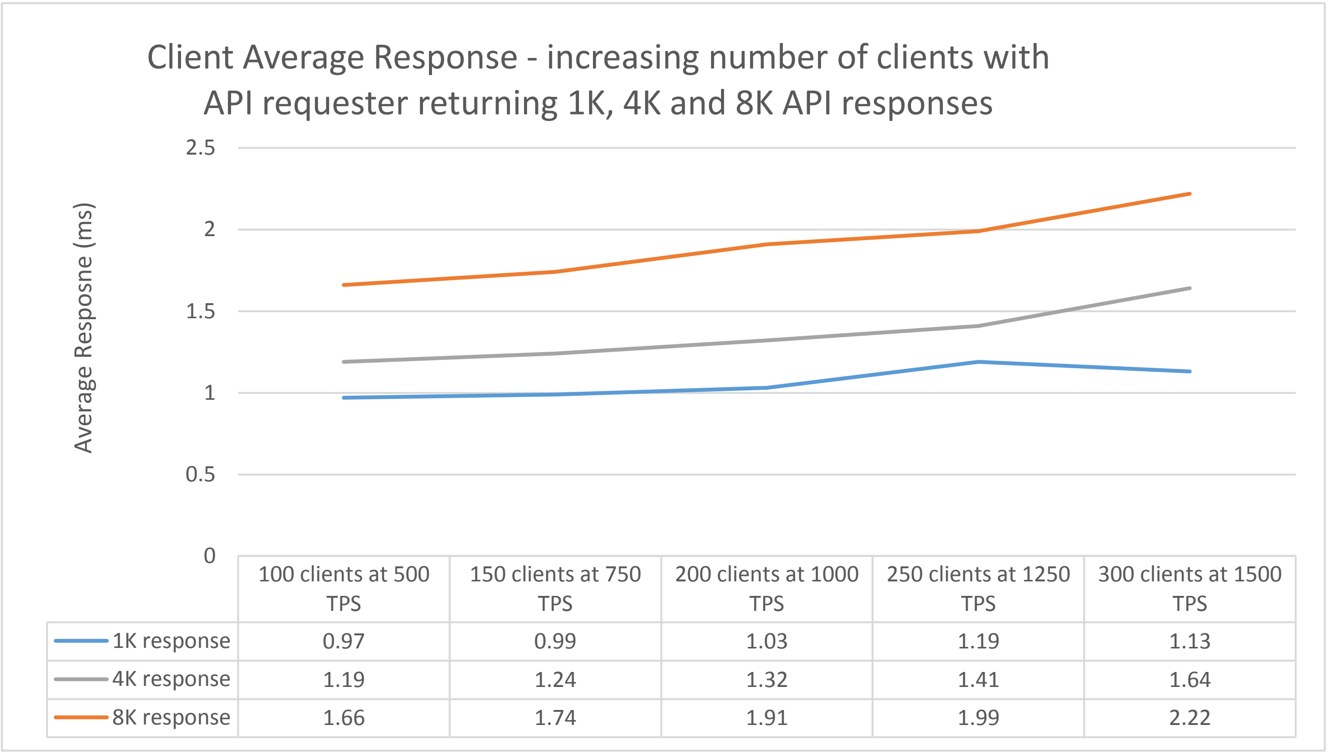
The transactions per second (TPS) for each run was maintained at the following rate regardless of payload size:

	100 clients	150 clients	200 clients	250 clients	300 clients
TPS:	500	750	1000	1250	1500

10. Client Average Response time

Ideally the average response time should be the same whether there is one client or 300 clients. Realistically, as the number of clients increase, the average response time can be expected to also increase due to resource contention in the system. Typically a maximum throughput in the system will be reached somewhere in any configuration at which point requests will begin to queue, thus increasing the average response time.

Figure 16 shows the client average response for 1K to 8K responses for different numbers of clients and transaction rates.



	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
1K response	0.97	0.99	1.03	1.19	1.13
4K response	1.19	1.24	1.32	1.41	1.64
8K response	1.66	1.74	1.91	1.99	2.22

Figure 16 Client Average Response for 1K to 8K responses with increasing numbers of clients and transaction rates

Observations

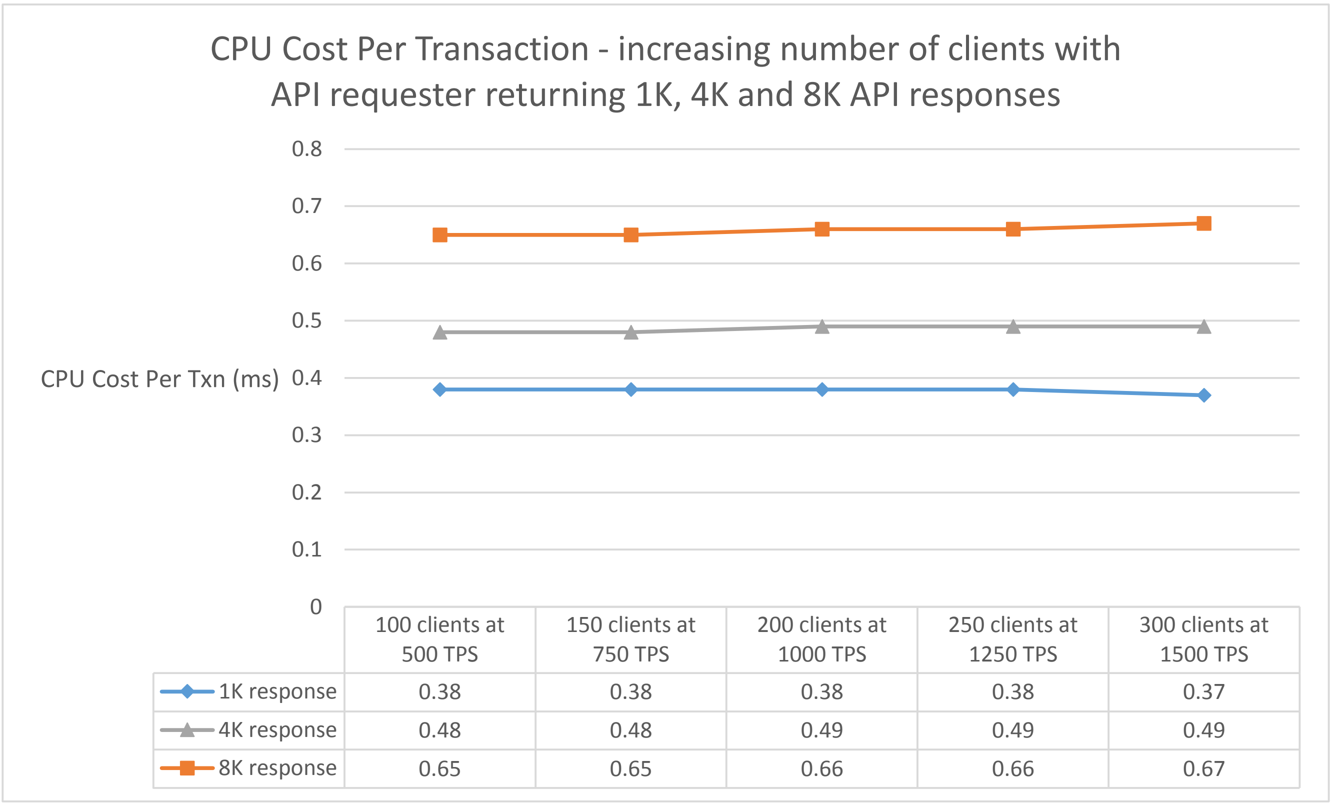
- ✓ z/OS Connect EE demonstrated acceptable scalability for average response times.
- ✓ Increasing the number of clients to run in parallel did not create unacceptable response times.

11. CPU Cost Per Transaction

The cost per transaction in CPU terms is measured in milliseconds (ms).

The calculation is made by dividing the CPU service time by the number of seconds in the SMF interval. This is then divided by the number of transactions per second (TPS) for the same SMF interval, and finally multiplied by one thousand to get the result in milliseconds.

Figure 17 shows the CPU cost per transaction for 1K to 8K responses for different numbers of clients and transaction rates.



	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
1K response	0.38	0.38	0.38	0.38	0.37
4K response	0.48	0.48	0.49	0.49	0.49
8K response	0.65	0.65	0.66	0.66	0.67

Figure 17 CPU cost per transaction for 1K to 8K responses with increasing numbers of clients and transaction rates

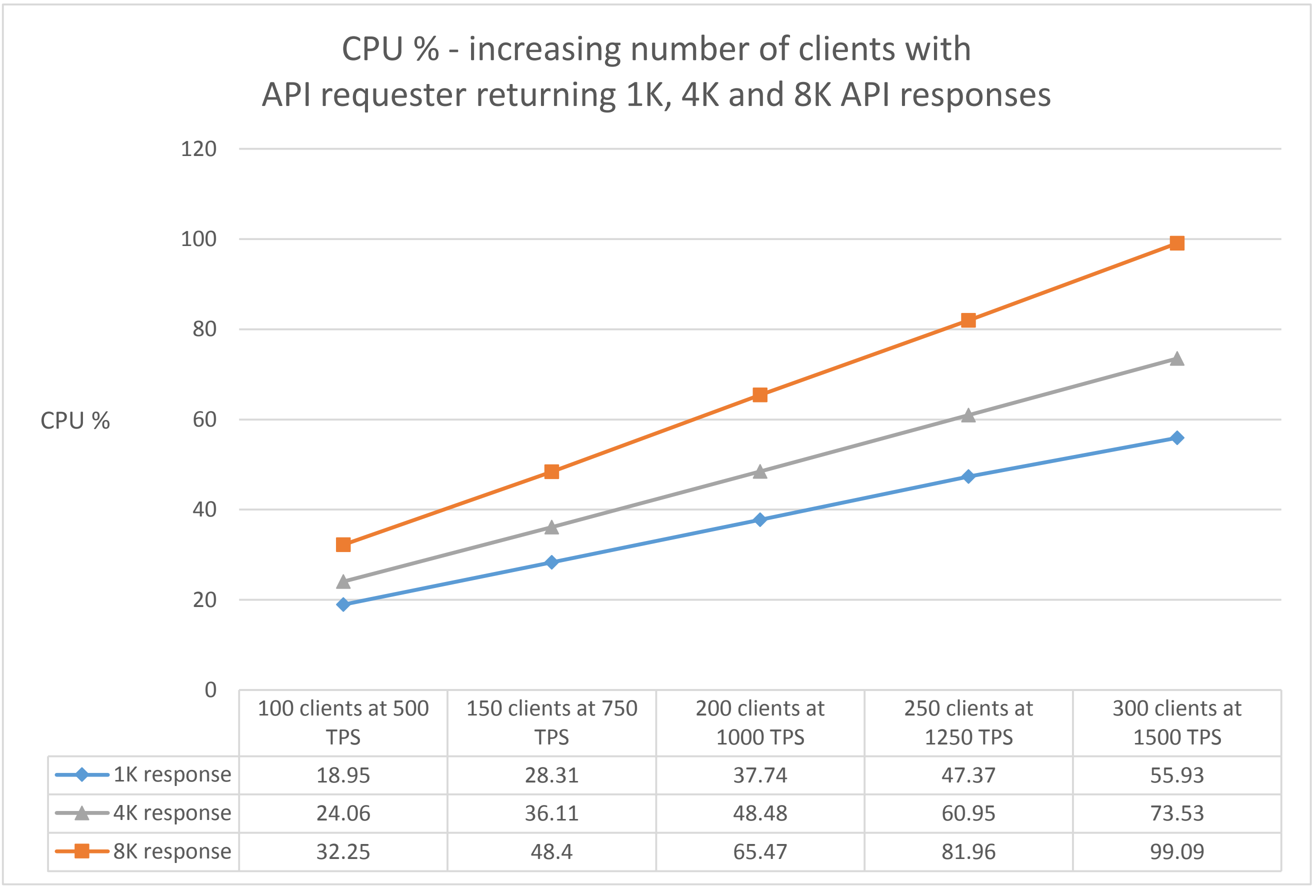
Observations

- ✓ z/OS Connect EE demonstrated good scalability with minimal increase in the CPU Cost Per Transaction as the number of clients increased.
- ✓ Increasing the number of clients to run in parallel from 100 to 300 clients resulted in little or no change in CPU Cost Per Transaction for this scenario.

12. CPU % usage

The CPU % usage will naturally increase as the number of clients running in parallel also increase. The CPU % includes all the GCPs (six in this environment), allowing a theoretical maximum of 600% for all products. This included the z/OS Connect EE server, Liberty server hosting the remote API, the client z/OS application, and the workload driver. In a typical customer environment these would not all run on the same LPAR.

Figure 18 shows the CPU % usage for the z/OS Connect EE server for 1K to 8K responses for different numbers of clients and transaction rates.



	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
1K response	18.95	28.31	37.74	47.37	55.93
4K response	24.06	36.11	48.48	60.95	73.53
8K response	32.25	48.4	65.47	81.96	99.09

Figure 18 CPU% usage for increasing number of clients and transaction rates

Observations

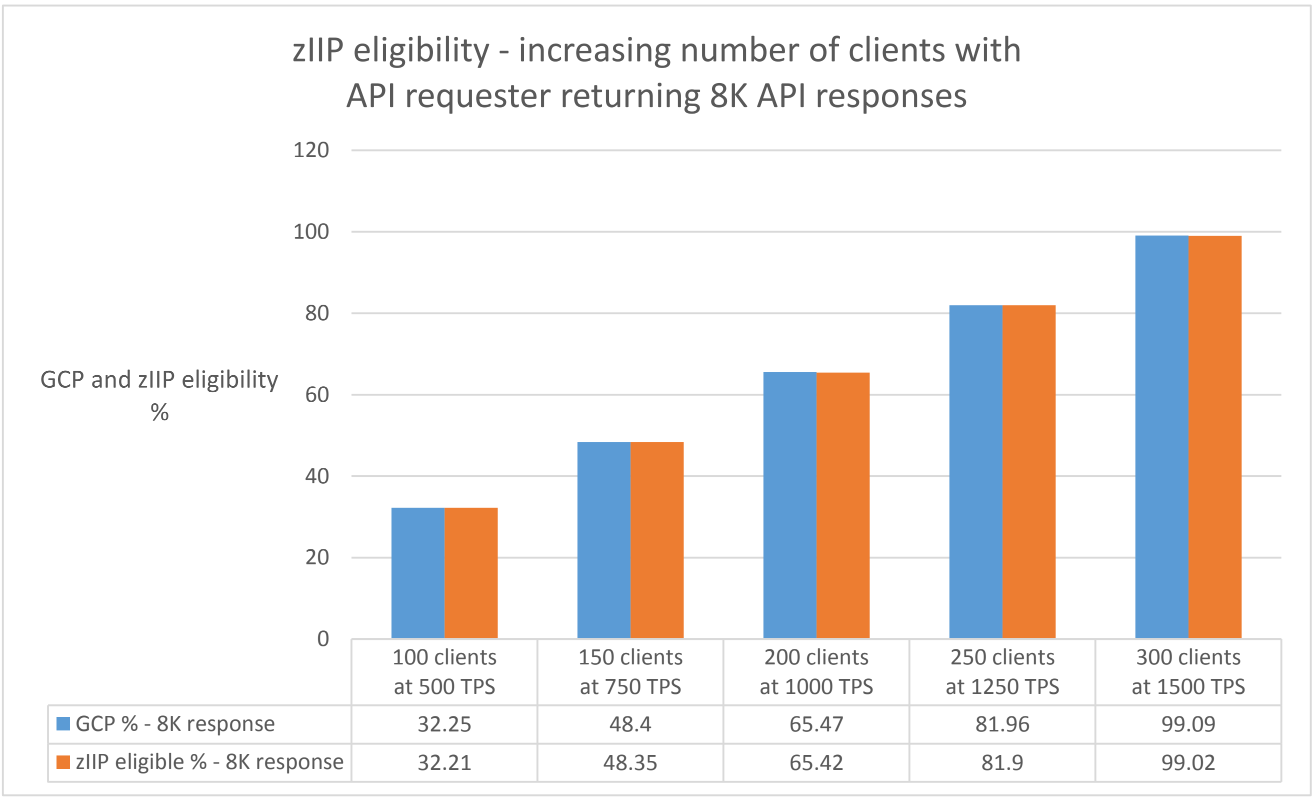
- ✓ z/OS Connect EE demonstrated good scalability.
- ✓ The CPU % usage increased linearly as the number of clients running in parallel were increased.

13. zIIP Eligibility

As z/OS Connect EE V3.0 is almost entirely written in Java the zIIP eligibility was observed to understand the benefits of offloading work to one or more zIIPs.

The environment is configured with six GCPs. Although zIIPs have not been used, zIIP eligibility was captured and is shown alongside the actual GCP usage.

Figure 19 shows the GCP % usage and zIIP eligibility for 8K responses.



	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
🟦 GCP % - 8K response	32.25	48.4	65.47	81.96	99.09
🟧 zIIP eligible % - 8K response	32.21	48.35	65.42	81.9	99.02

Figure 19 GCP CPU % and zIIP eligibility for increasing numbers of clients with 8K responses

Observations

- ✓ z/OS Connect EE is a Java-based product and so over 99% of the product is eligible to be offloaded to zIIP.
- ✓ The potential usage of zIIP processors scaled well with the increasing numbers of clients.
- ✓ Although not shown, smaller payloads exhibited the same results in that approximately 99% CPU is offloadable to zIIP.

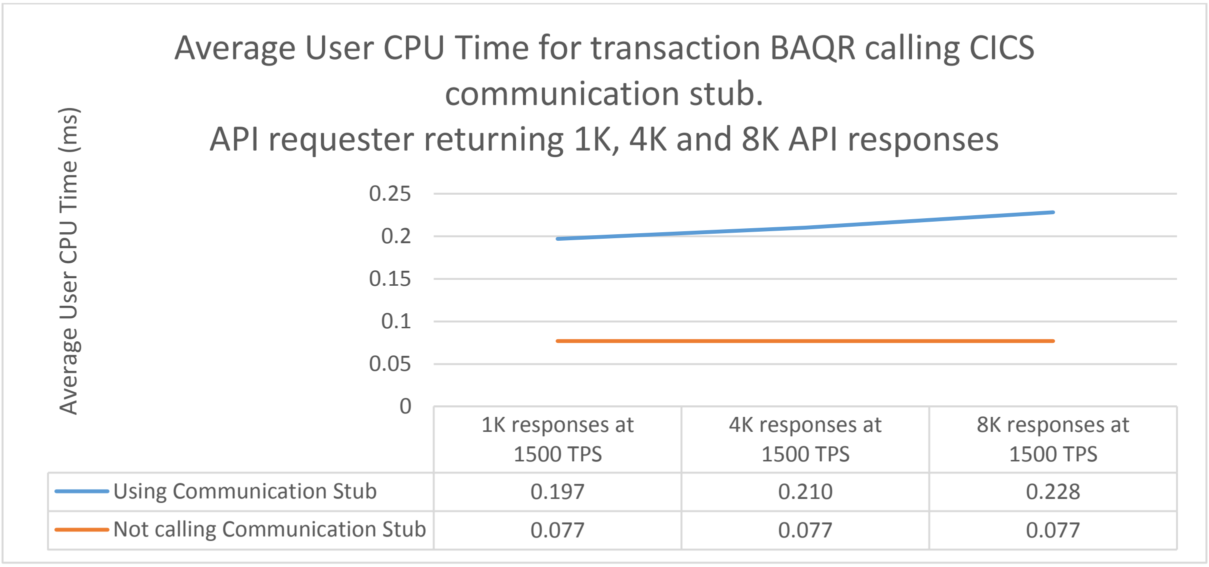
14. Communication Stub

The z/OS application used in this report used the CICS communication stub to communicate requests and responses between the CICS application and the API requester within z/OS Connect EE. However, it could equally have been the communication stub used for IMS and non-CICS z/OS applications.

To calculate the cost of the communication stub, measurements were taken on CICS. To do this, CICS statistics was enabled to gather data of the CICS communication stub and the average CPU time was obtained using CICS Performance Analyser.

Performance runs using 300 clients at 1500 transactions per second sent requests that received 1K, 4K and 8K response payloads from the remote API.

Figure 20 shows the average user CPU time for transaction BAQR:



	1K responses at 1500 TPS	4K responses at 1500 TPS	8K responses at 1500 TPS
Using Communication Stub	0.197	0.210	0.228
Not calling Communication Stub	0.077	0.077	0.077

Figure 20 Average User CPU Time for transaction BAQR

Observations

- ✓ The CICS communication stub demonstrated good scalability.
- ✓ The average user CPU time for transaction BAQR increased linearly as the size of payload increased.

15. Conclusions

The following conclusions can be drawn from this report:

1. A single z/OS Connect EE V3.0 server managing requests from a z/OS application to call remote APIs demonstrated good scalability (in terms of number of clients and payload size) for
   - ✓ Transactions Per Second
   - ✓ Average Response time of requests
   - ✓ CPU Cost Per Transaction
   - ✓ CPU % Usage
   - ✓ Potential zIIP offload.
2. The API requester scaled well.
3. z/OS Connect EE V3.0 is almost entirely written in Java and so would benefit from offloading work to one or more zIIPs.
4. The z/OS Connect EE communication stub scaled well.

Note:

1. The payload sizes for the responses in this report are up to 8K.
2. Analysis of other payload sizes, or different hardware, have not been completed at this time, so there is no guarantee that equivalent observations will be seen in other configurations.
3. Due to the effects on system performance of machine hardware, levels of software configuration and payload, equivalent observations might not be seen on other systems.