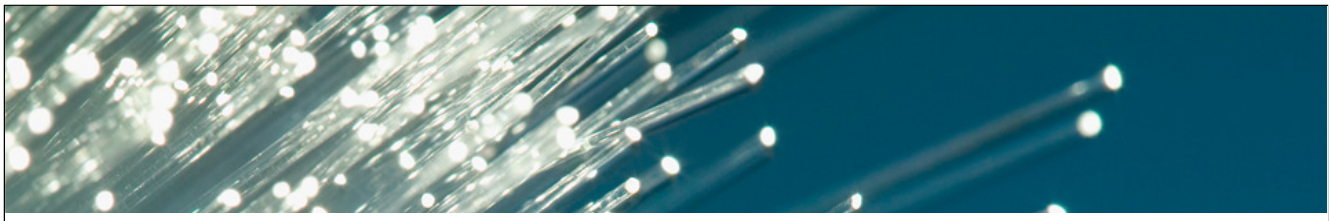
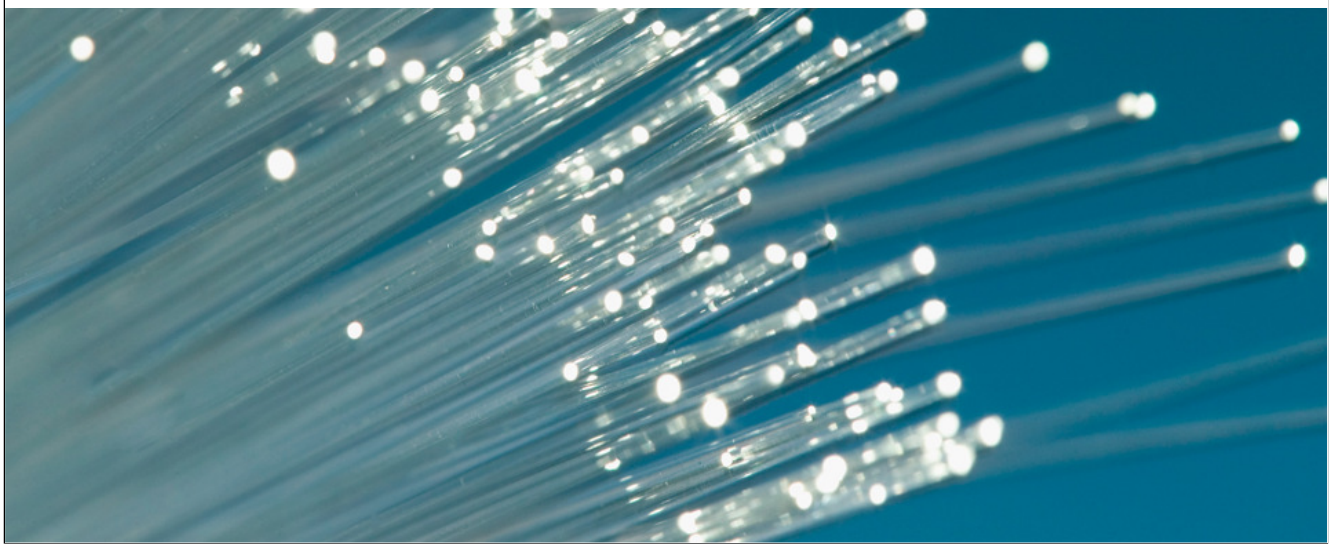


IBM zEnterprise

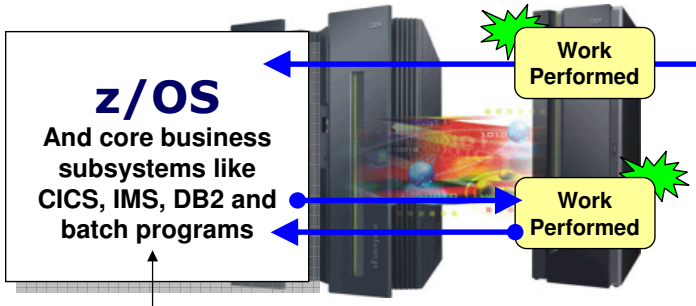
zEnteprise Usage Scenarios



This page intentionally left blank

Outside-In … and Inside-Out

Think of yourself as z/OS on the z196 as the center of zEnterprise. Then think about where work might originate:



Outside In

Work originates from the outside

zBX serves to handle work prior to arrival at z/OS

Examples: application, web and edge-serving functions

Inside Out

Work originates on z/OS

zBX serves to handle certain work on behalf of mainline process

Examples:

- *Query Optimizers*
- *SAS® or SPSS*
- *R statistical programs*
- *PDF and graphics generation*
- *Bulk data encryption*

Use z/OS cycles for work best suited for that platform

Offload portions but maintain operational context as part of z/OS

3

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

If you begin with the z/OS system being the center of the work, then there are two essential categories of workload architecture:

Outside In – this is where the work originates outside the z/OS system and propagates in to the z/OS where key data and transaction systems such as DB2, CICS and IMS reside. This is a very common architectural approach used by web facing solutions. In this case the zBX could serve as the hosting technology for functions such as HTTP servers or caching servers. This would be "edge serving" with a very close-proximity edge.

Inside Out – this is where work starts on z/OS but some portion of it is offloaded to other processing systems. The reason this might be considered is because some portion of the work can't be done on z/OS (this is very rare) or, more likely, some portion of the work simply makes better sense to process on a different platform. Examples of this are query optimizers, the use of SAS®, SPSS or other statistical analysis programs written with R. Finally, the example we'll show in lab is the generation of PDFs as part of a batch process.

It's no secret that z/OS cycles are considered a valuable resource. So it makes sense to use those cycles as wisely as possible. At the same time, however, it's important to maintain the operational control and context of the job as it starts and completes on z/OS. This includes integration with enterprise schedulers, automation tools, and maintaining job output – every part of the job output – within JES spools.

We'll focus on the "inside out" pattern in this unit. And our focus will be on an enabling technology from Dovetail Technologies called "co:Z".

Introducing co:Z

co:Z is no-fee technology suite from Dovetail Technologies (originators of JZOS). There are several key pieces of this technology:

Our flagship product, the **Co:Z Co-Processing Toolkit for z/OS** was introduced in 2006 and is used widely throughout the z/OS community. Our users have discovered that Co:Z offers them the essential tools they need to connect their z/OS platforms to other computing environments securely and reliably. Browse Co:Z's capabilities by visiting the product links below, then try out and use the product **free**.

- **Co:Z Launcher** - Cooperative processing
- **Co:Z Dataset Pipes** - Dataset conversion
- **Co:Z SFTP** - File transfer
- **Co:Z Batch** - Better BPXBATCH

co:Z Launcher

A set of z/OS functions and utilities that allows z/OS and non-z/OS platforms to work more cooperatively

co:Z Dataset Pipes

A set of z/OS and non-z/OS utilities that make streaming data between platforms much easier

http://dovetail.com/solutions.html

4

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

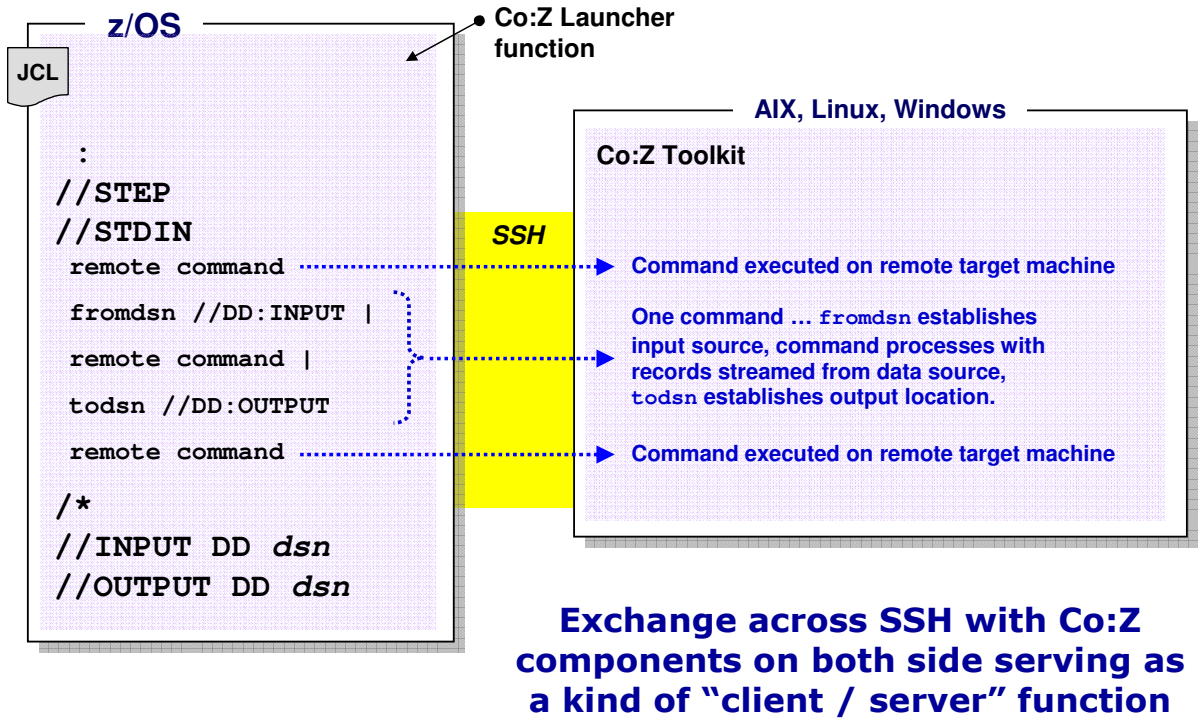
The co:Z product is a set of technologies that allows z/OS work to cooperatively work with work performed on non-z/OS platforms. In the upcoming lab you'll use co:Z to have a batch job run on z/OS with a portion of that work being done on an AIX virtual server.

The Dovetail website has a wealth of information on this product. We will focus our attention on the Launcher and Dataset Pipes.

Think of the co:Z launcher as z/OS program that enables the execution on remote platforms, and Dataset pipes as a set of functions that makes access to and from z/OS datasets considerably easier. For example, if you wanted the program on the remote platform to access a member of a PDS, the use of Dataset pipes makes this as easy as naming the PDS. The rest of the exchange, including code page conversion and record handling, is provided by the co:Z product.

SSH Pipe with Handlers on Both Sides

This illustrates how Co:Z works at a high level:



Exchange across SSH with Co:Z components on both side serving as a kind of "client / server" function

5

This chart is illustrating how Co:Z works between z/OS and the target machine. In our workshop the target machine is AIX, but Co:Z has support for Linux and Windows as well.

Co:Z activity is initiated from the z/OS side of the equation. It uses an SSH pipe it establishes to the target machine to pass the commands. We'll see how Co:Z knows which target server in a few charts. The commands that get passed to the target and executed there are enclosed within the `STDIN` of the JCL job.

In the abstract example above we see that some remote command is coded in the `STDIN` section. Co:Z flows that command across SSH to the target, where the Co:Z component on the target receives the command and executes it.

The next three lines in the chart are actually one command. Note the continuation vertical bars. This one command does three things

- **`fromdsn`** is a Co:Z keyword command that allows an input source on z/OS to be established using normal JCL DD processing. That input data source may be a USS file, a sequential data set or a partitioned data set.
- Some command is executed that uses fromdsn as the input source. Records are streamed across the SSH pipe. The Co:Z data set pipes function provides support for complexities such as streaming record by record, and code page conversion issues.
- **`todsn`** is another Co:Z keyword command that does the reverse of `fromdsn`. It establishes the output data location. Here again, that's on z/OS and resolved using standard JCL DD. Records are streamed *back* to z/OS and to the data set (or USS file).

The example shows another remote command just to illustrate that the `STDIN DD` commands may be quite lengthy and do many separate discrete things. The `INPUT DD` and `OUTPUT DD` cards are shown at the bottom.

JCL Straight from the Upcoming Lab

So you see how easy this can be …

```
//ZMGRT#B   JOB ...
//PROCLIB JCLLIB ORDER='SYS1.COZ.CNTL'
//RUNCOZ EXEC PROC=COZPROC,ARGS='zmgrt#@zmgrt#c.dmz'
//STDIN DD *
echo "We are running on: " `uname -sn`
fromdsn //DD:INPUT |
sed '/\*/d' |
todsn //DD:OUTPUT
/*
//INPUT  DD DSN=ZMGRT#.JCL.CNTL(TEXTIN),DISP=SHR
//OUTPUT DD DSN=ZMGRT#.JCL.CNTL(TEXTOUT),DISP=SHR
//*
```

Executed on remote and returned to z/OS

```
 This is line one
 This is line two
*Comment Line
 This is line three
*Comment Line
 This is line four
```

```
This is line one
This is line two
This is line three
This is line four
```

Defined and submitted on z/OS

Executed on AIX

Results streamed back to z/OS

Simplistic? Yes, but it shows the potential of this technology

6

Here's an example of JCL that uses co:Z. This is straight from the upcoming lab.

Notice that the EXEC statement is calling a co:Z proc, where a co:Z module will be executed. One of the arguments passed in is the ID and host of the target system to which you wish co:Z to connect.

The input is provided in STDIN. The first command that would be executed on the target system is a simple echo command where 'uname –sn' is executed. This returns the operating system name and the ID being used on the target system. We have that in the lab to "prove" it's being run on the target AIX system.

The input and output data definitions are provided using the dataset pipes commands fromdsn and todsn. Those are functions that come with the co:Z product. Notice how the data sets are defined in line – as DD statements. The DD is resolved in JCL after STDIN. In this case the input and output is a FB 80 PDS.

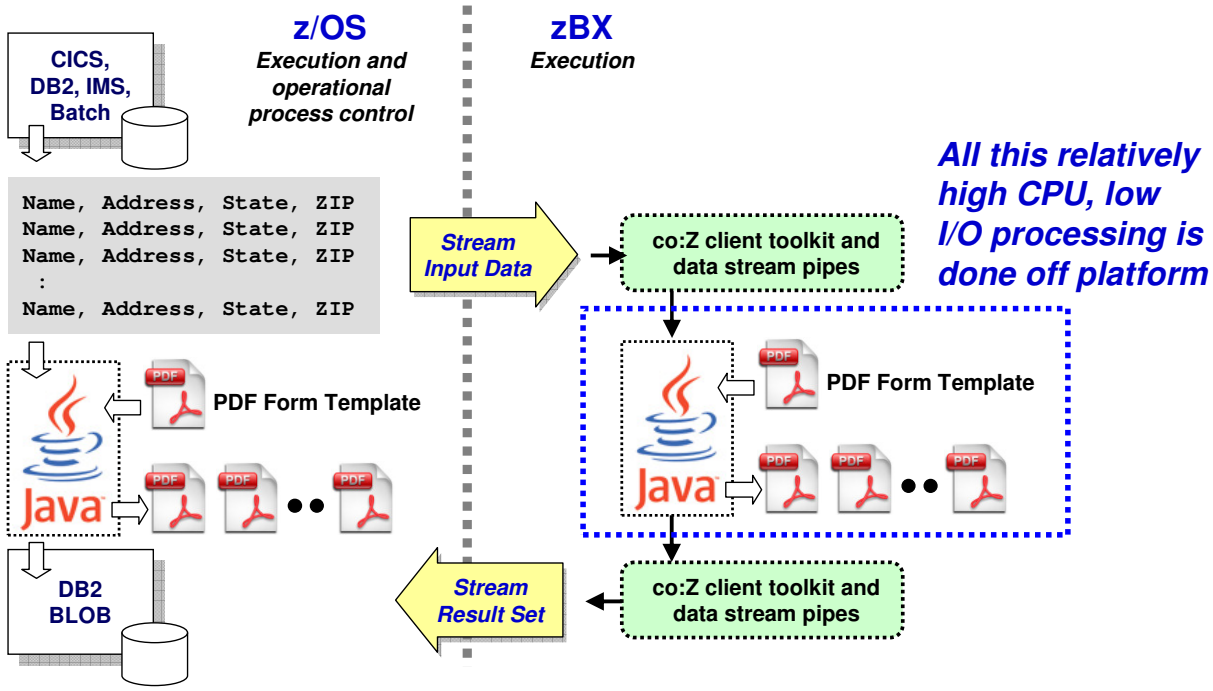
The lab is a very simple one that reads the data and strips out comment lines, then puts the results back on z/OS. The process that does that stripping is "sed" that you see on the chart.

The processing is done on AIX but all the data and the job execution definitions are on z/OS. Co:Z on z/OS opens up SSH to the target server and processes the commands in STDIN. The co:Z dataset pipes function assists with the streaming of data from the dataset on z/OS to the AIX environment where the processing is done. Dataset pipe function is used to stream the results back to z/OS.

Is this simplistic? Yes, very much so. But it helps illustrate the potential of this technology to do cooperative off-board processing with operational control being maintained at the z/OS level.

A Real World Example …

Imagine a large extraction of data that needs to be populated into customized PDF files … one PDF per record extracted from database



7

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

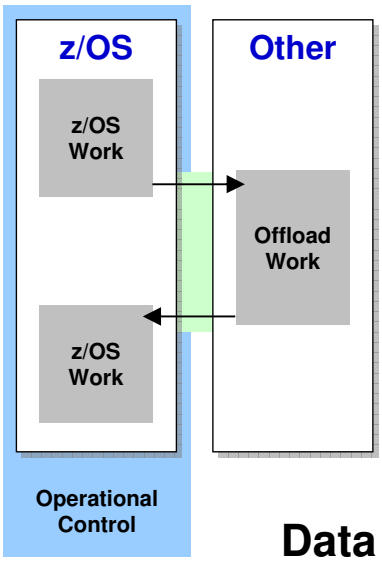
This is a logical portrayal of another portion of the upcoming lab. This is a business challenge faced by a customer who wished to do mass mailings of PDF files based on address information extracted from their z/OS data systems.

PDF generation *can* be done on z/OS. It's a relatively high processing effort without a lot of I/O, except for accessing and reading the address extract file. The desire was to run this PDF processing off platform but to do so within the operational context of the submitted batch job that did the data extraction.

The solution created uses co:Z to accomplish this. The PDF creation program (Java) runs off platform with the data input file and the produced PDF being streamed using co:Z dataset pipes.

Other Possible Uses

Just a few other examples we've seen discussed as potential uses of this remote execution technology:



All three are statistical analysis products or programming languages

Statistical analysis tends to be CPU intensive with relatively little I/O

While they *can* be run on z/OS it is often seen as a less-than-optimum use of z/OS resources … hence offload

Data maintained and generated on z/OS … results returned to z/OS … offload performed within the context of the z/OS jobs

8

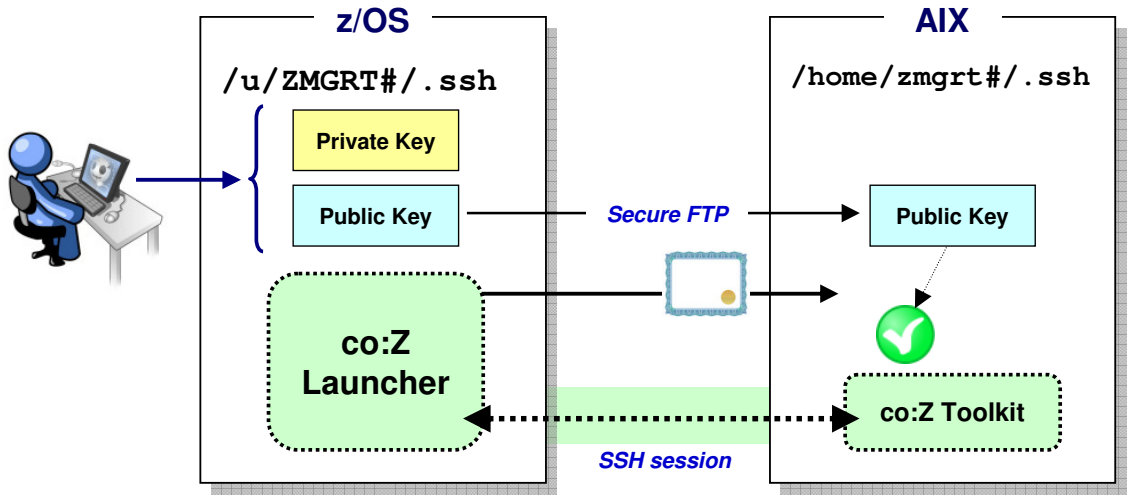
The use of statistical processing is another potential use of co:Z as part of an "inside out" model of procressing on the zBX. Examples of such statistical analysis programs include SAS, SPSS or programs written with R.

The concept is the same in all cases … the job is run as a standard z/OS batch job; the operational context is maintained on z/OS; the data is streamed to the off-board platform and streamed back; data is not left on the target platform.

Again, in these cases statistical analysis programs could be run on z/OS … but there may well be good business reasons to run this portion elsewhere. This kind of processing tends to be fairly CPU intensive. If the available CPU on z/OS is seen to better be used for other processing, then offloading like this serves the business better.

co:Z and Authentication

As a preview of the upcoming lab … a setup activity involves generating and propagating security certificates so no password prompts required



Not unique to co:Z … the use of certificates to assert identity is a common practice. This avoids having to answer ID and password prompts out of your batch program using co:Z

9

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

We have already established that co:Z makes extensive use of SSH to stream the data back and forth between z/OS and the target system. SSH requires authentication. But having IDs and passwords embedded in STDIN processing is not desirable. Co:Z gives you the opportunity to use certificates to authenticate with the target system and not require answering password prompts.

This involves generating a set of keys on z/OS and sending the public key to the target system. Then when co:Z on z/OS opens the SSH session to the target and presents the certificate the target system can know it can trust the request and allow the SSH session to be run under a specific identity.

This is what you'll do in lab. We're covering that here to have some background on what the lab steps are doing and why.

The use of certificates for authentication is actually quite common. It's not unique to co:Z. But it is possible to use it with co:Z, and that's what you'll do in the upcoming lab.

co:Z and Encryption

co:Z has a configurable option to flow `fromdsn` and `todsn` traffic over an open socket … thus avoiding encryption

Your JCL Library and JOB

```
//ZMGRT#B   JOB ...
//PROCLIB JCLLIB ORDER='SYS1.COZ.CNTL'
//RUNCOZ EXEC PROC=COZPROC,ARGS='zmgrt#@zmgrt#c.dmz'
//STDIN DD *
  :
```

SYS1.COZ.CNTL(COZPROC)

```
//EXCOZ    PROC ARGS=,
//   LIBRARY='SYS1.COZ.LIBRARY',
//   COZCFGD='SYS1.COZ.CNTL(COZCFGD)',
//   REGSIZE='
 :
```

SYS1.COZ.CNTL(COZCFGD)

```
########################################
# Co:Z Installation specific settings
########################################
server-ports=8040-8048
ssh-tunnel=true
#ssh-tunnel=false
```

Default, encryption

Option, no encryption

TRUE … then target program connects to z/OS over encrypted SSH tunnel

FALSE … then target program connects directly … no tunnelling, no encryption

What is your policy with regard to security of the IEDN?

10

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

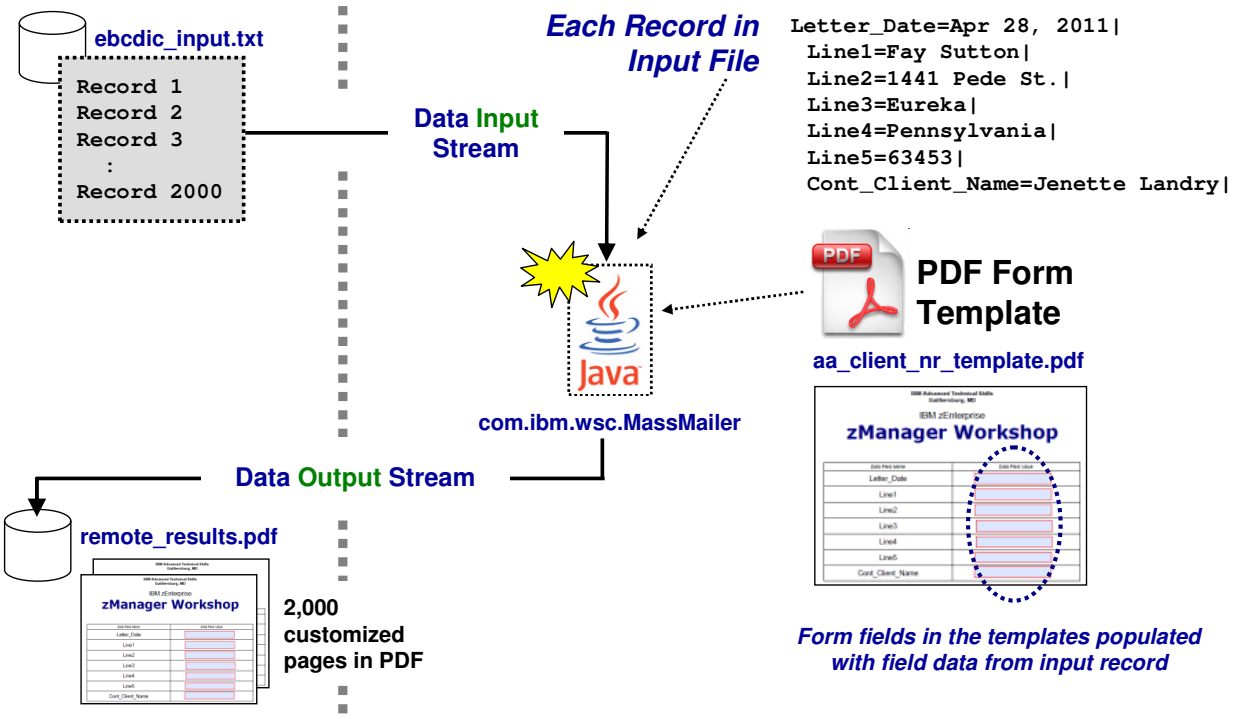
In addition to authenticating with certificates, it's also possible to avoid SSH encryption of data with co:Z. Co:Z does this by providing a way to establish a clear socket connection between z/OS and the target. This is done with the "ssh-tunnel" configuration paramater. The default is "true," which means fromdsn and todsn processing is done over encrypted SSH. Setting this value to "false" is a way to avoid encryption.

Why would you do that? Because encryption represents a fraction of additional processing effort you may deem unnecessary. If the amount of data being processed back and forth is significant then the savings by not encrypting may be worth pursuing.

The question then becomes a security policy one … given the nature of the IEDN as a flat network within the zEnterprise, do you believe it is sufficiently secure to allow for data encryption to be avoided? This is a question that should be considered in light of what the IEDN provides and how, given proper security discipline, can be as secure as any private LAN in the datacenter. If it is deemed to be secure then the avoidance of encryption provides an additional benefit of the design of the zEnterprise.

Background on the PDF Generation Lab, Part 1

So that it makes better sense when you perform the lab steps …



11

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

Just a little background on the upcoming lab …

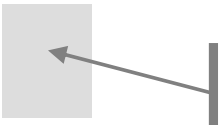
We will supply a flat delimited file with addresses. Each record has seven input fields delimited with a vertical bar. This file represents the data extraction of a set of names and addresses.

The co:Z job you will submit will run a Java program on the AIX virtual server and use fromdsn to stream the input file from its location on z/OS. The template PDF file is read into the Java program and its input fields populated with the data in the input record … one input record per PDF letter generated, all concatenated into a single results PDF. It uses todsn to pipe the created PDF file – all 2,000 pages of it – back to z/OS.

Background on the PDF Generation Lab, Part 2

The JCL you will submit on z/OS that initiates and controls remote execution:

```
//ZMGRT#A   JOB (D999,MISC),'JOB RUNS ON AIX',
//             MSGLEVEL=(1,1),
//             REGION=0M,
//             CLASS=A,
//             MSGCLASS=O TYPRUN=SCAN
//PROCLIB JCLLIB ORDER='SYS1.COZ.CNTL'
//RUNCOZ EXEC PROC=MCC#COZP,ARGS='zmgrt#@zmgrt#c.dmz'
//STDIN DD *
```



Issues `uname -sn` to "prove" running on AIX

Sets up the dataset pipes (called "FIFOs")

Invokes the shell script which launches Java and runs the `com.ibm.wsc.MassMailer` program

Cleanup: flushes FIFOs, kills co:Z client child processes, removes temporary FIFO files

```
/*
```

12

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

Here's a brief illustration of the JCL you'll submit on z/OS. A key piece of this is the arguments passed into the process. It takes the form of the host name of the target to connect to, and the ID to use when connecting. The chart is showing the use of your team's ID and co:Z virtual server. In this case "#" is meant as a place where you substitute in your team number.

The instructions that get executed on the target AIX system are contained in the STDIN processing. The chart shows the basic steps that are executed.

Background on the PDF Generation Lab, Part 3

For good measure we'll have you run another job[1] that runs the Java/PDF processing locally on z/OS. Then you'll compare the results.

Job Statistics:

```
+- Step Statistics - WSCACTRT 3.3 ---------------------------------- HBB7770 -+
|JOB(ZMGRT#Y ) Step#  1 (JAVA     .JAVAJVM ) Pgm(JVMLDM50) - Return Code    00 |
+-------------------------------------------------------------------------------+
|Elapsed Time: 00:00:xx.xx  CPU Time(TCB): 00:00:xx.xx  (SRB): 00:00:00.00      |
|RCT CPU Time: None         I/O Int. Time: None         ICSF Count: None       |
|Service Units: CPU=     1,306K SRB=        81  I/O=     77,007  MSO=         0 |
|Transactions: ended:None   Active for: 00:00:18.68 Resident for: 00:00:18.68 |
```

Lab Write-Up:

Job	*Elapsed Time*	*CPU Time*
PDF on P		
PDF on Z		
Delta		

What you'll see: **Elapsed time relatively close to one another** **CPU time much less when executed remotely**

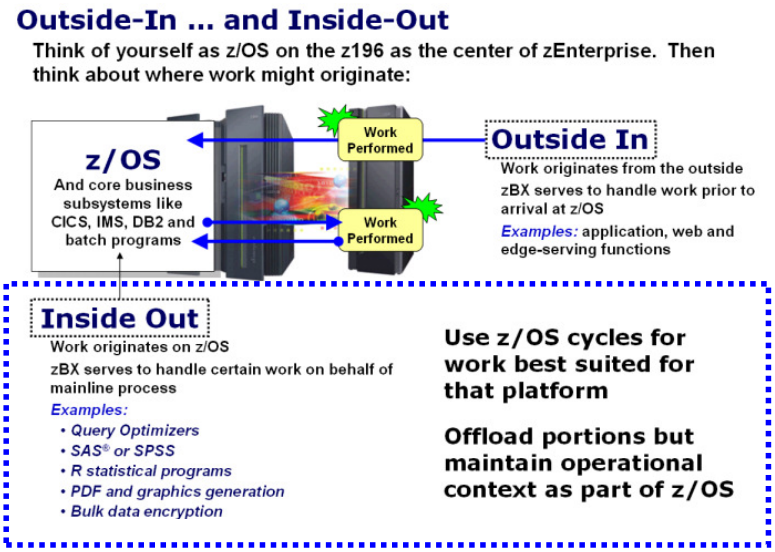
[1] **The local execution uses the same input file and Java program, but not co:Z launcher – uses JZOS and local access to the files**

13

Note: this chart uses animation. The message is enhanced when seen with animation in motion.

In addition to running the PDF generation on the remote AIX virtual server, we will also have you run a local PDF creation process. This is so you may then compare the elapsed and CPU time of each. You will see that using co:Z to offboard the processing does not take too much additional elapsed time, and it uses virtually no CPU for the remote execution.

Summary



Focus of this unit

Objective was to get us thinking about "inside out" processing and what other uses we can find for zBX as an extension of core z/OS processing

And our summary.

End of Unit