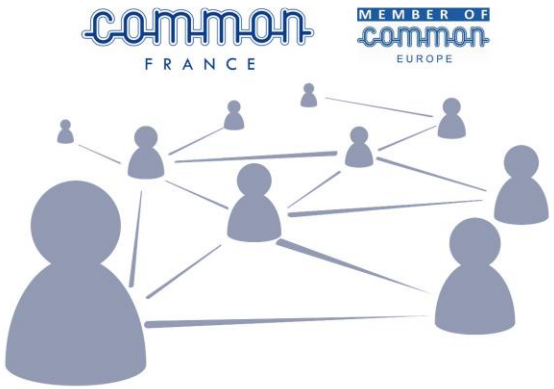
Integrating OpenSource with your current Technology Stack

Koen Decorte



Qui sommes nous?

- IBM i ISV et IBM business partner situé à Anvers, Belgique et à Madrid Espagne.
- Plus de 40 ans d'éxpérience avec IBM i et ses prédécesseurs.
- Applications : comptabilité, OCR, query tool, MES, ERP
- Expertise en RPG, SQL, PHP, HTML, Unity, nodejs, linux…
- Site web : www.cdinvest.eu
- IBM Champion depuis 2018 et membre CEAC, Trésorier Common Belgique
- **What others talk about, we do.**

Nos Case studies

- JORI : https://www.ibm.com/case-studies/jori
- Fibrocit : https://www.ibm.com/case-studies/fibrocit-systems-furniture-design
- Cras : https://www.ibm.com/case-studies/cras-systems-open-source
- Oris : https://www.ibm.com/case-studies/ORIS
- Deknudt Frames : https://www.ibm.com/case-studies/deknudt-frames
- Bonehill : https://www.ibm.com/case-studies/immo-bonehill-systems-hardware-website-compliance
- Winsol : https://www.ibm.com/case-studies/winsol-systems-hardware-manufacturing-digitization
- Vanmaele : https://www.ibm.com/case-studies/wijnen-van-maele-systems-software-ibm-i
- Steffimmo : https://www.ibm.com/case-studies/steffimmo-systems-software-property-maintenance
- CSM : https://www.ibm.com/case-studies/csm-bvba-systems-hardware-trade-power-i
- Stonetales : https://cms.ibm.com/case-studies/stonetales-properties-power-upgrade
- ID logistics : https://www.ibm.com/it-infrastructure/us-en/resources/power/ibm-i-customer-stories/#/id-logistics/

Agenda

- Introduction aux PASE
- Configuration de l’environnement PASE
- Qu’est qu’un Shell
- XMLSERVICE interaction

Agenda

- Nodejs, PHP et Python et ILE
- L’IBM i Toolkit
- L’outil Unixcmd
- RPG et Pase – nos techniques et astuces

Open Source Examples to use

- Curl
- ImageMagick (convert/mogrify/identify)
- Ghostscript (gs)
- Python (python)
- Tesseract (OCR/Gvision)
- ….

All these tools use a shell command so you need to interact with it !

PASE

Vue d’ensemble de Pase

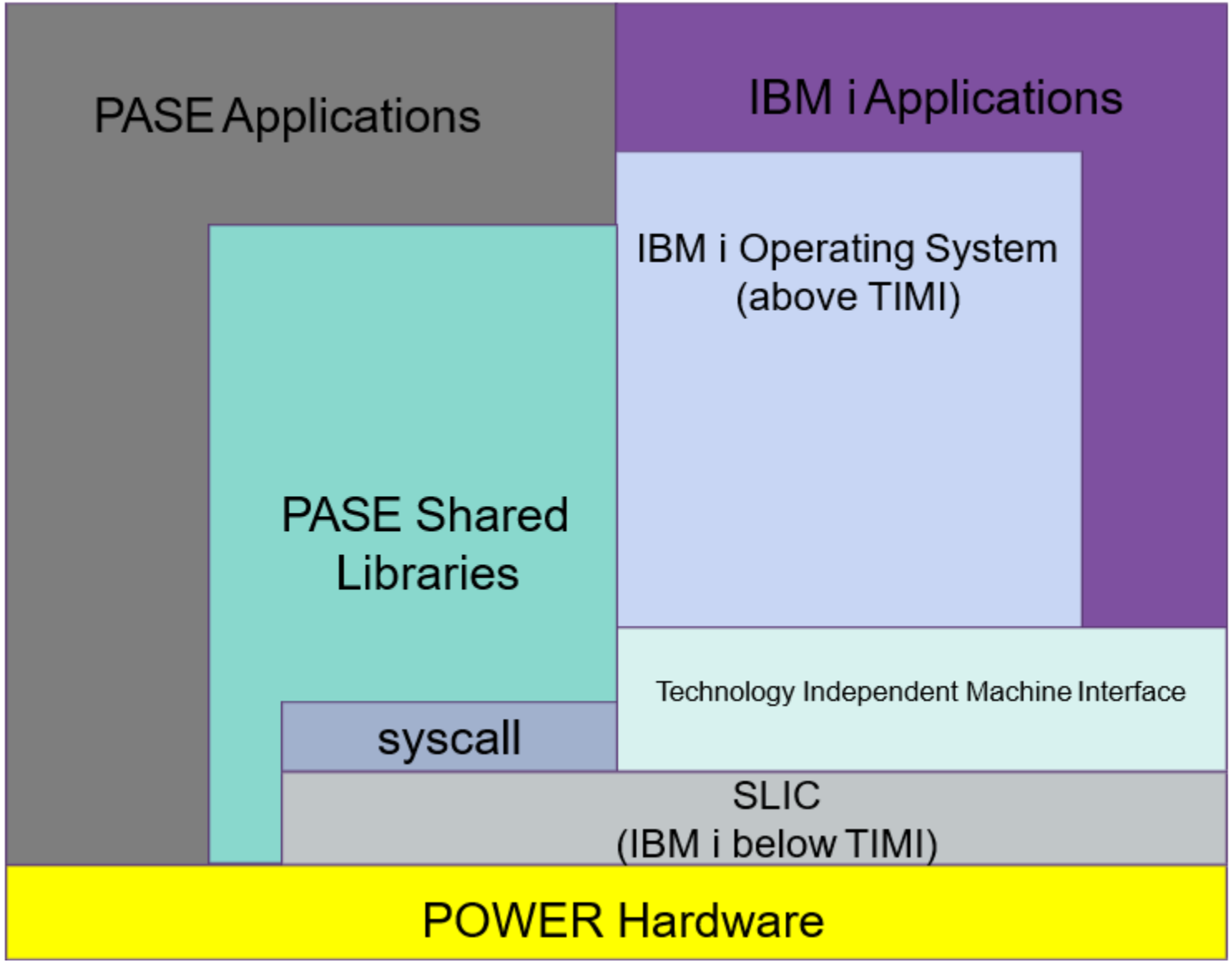
- En 2000, AIX et OS/400 pouvaient fonctionner sur les mêmes processeurs POWER.
- Cela a créé la possibilité pour les exécutables basés sur MI et AIX de s’exécuter sur le même hardware et dans la même partition
- PASE permet à ces binaires de s’exécuter dans le même processus
- PASE n’est pas une version d’AIX mais plutôt un ensemble de bibliothèques AIX
  - Adapté pour communiquer avec SLIC plutôt que directement avec le kernel AIX
- PASE obtient la mémoire des mêmes pools de teraspace SLIC utilisés par ILE
  - Pour le “program run stack, heap, and shared memory” PASE peut seulement voir la mémoire que PASE a obtenu par ses propres syscall APIs

Ce que PASE n’est pas

- PASE n’est pas un environnement émulé
- PASE n’est pas un environnement distinct d’IBM I
  - À titre d’exemple, le même système de fichiers intégré (IFS) est accessible à partir de PASE que de toute autre partie d’IBM I
  - Un autre exemple, les applications/outils/programmes démarrés dans PASE peuvent accéder aux données Db2 et aux programmes ILE

Architecture de Pase

- PASE fournit un ensemble de bibliothèques partagées AIX qui s’exécutent directement sur le processeur POWER
  - Les applications dans PASE bénéficient des mêmes performances que les applications exécutées dans une partition AIX
- Une interface syscall permet aux applications dans PASE d’appeler des applications ILE et d’accéder aux données Db2


PASE Applications
IBM i Applications
IBM i Operating System
(above TIMI)
PASE Shared
Libraries
Technology Independent Machine Interface
syscall
SLIC
(IBM i below TIMI)
POWER Hardware

IBM speak

- PASE supports the application binary interface (ABI) of AIX and provides a broad subset of the support provided by AIX shared libraries, shells, and utilities.
- PASE also supports the direct processing of IBM PowerPC machine instructions, so it does not have the drawbacks of an environment that only emulates the machine instructions
- PASE applications:
  - Can be written in C, C++, Fortran, or PowerPC assembler
  - Use the same binary executable format as AIX PowerPC applications
  - Run in an IBM i job
  - Use IBM i system functions, such as file systems, security, and sockets

https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzalf/rzalfwhatispase.htm

IBM speak

- PASE run-time runs on the Licensed Internal Code (LIC) kernel on the IBM i operating system.
- The system provides integration of many common IBM i functions across PASE and other runtime environments including Integrated Language Environment (ILE) and Java.
- PASE implements a broad subset of AIX system calls
- System support for PASE enforces system security and integrity by controlling what memory a PASE program can access and restricting the program to use only unprivileged machine instructions

https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzalf/rzalfwhatispase.htm

Intégration – aller d’ici à là (et vice versa)

- Call PASE de l’IBM i
  - `QSH CMD('ls /home/KOEN’)`
- Call IBM i à partir de PASE
  - `system "WRKOBJLCK OBJ(MYFILE) OBJTYPE(*FILE)"`
- Des intégrations plus robustes telles que l’accès aux programmes ILE et à partir de langages open source sont possibles – nous y reviendrons plus tard

Configuration de l’écosystème OSS dans PASE

Vue d’ensemble de RPM

- Cela permet l’installation de packages open source de manière Linux avec PASE
- La pile RPM contient de nombreux packages, notamment:
  - Python
  - Node.js
  - The 'less' utility
  - Git
  - The 'updated' and 'locate' utilities
  - GCC and other development tools
  - GNU Nano
  - Autres…

https://bitbucket.org/ibmi/opensource/src/master/

Les RPM ne sont pas des RPM AIX. Ce sont des RPM IBM i uniquement pour le logiciel IBM i. Construit sur IBM i, pour IBM i.

Installer RPM/YUM

- Step 1: Download the bootstrap file to your PC:
  - ftp://public.dhe.ibm.com/software/ibmi/products/pase/rpms/bootstrap.sql
- Step 2: Run the SQL script against the system you want to install RPMs on

```
create or replace table qtemp.ftpcmd(cmd char(240)) on replace delete rows;
create or replace table qtemp.ftplog(line char(240)) on replace delete rows;

insert into qtemp.ftpcmd(CMD) values
   ('anonymous anonymous@example.com')
  ,('namefmt 1')
  ,('lcd /tmp')
  ,('cd /software/ibmi/products/pase/rpms')
  ,('bin')
  ,('get README.md (replace')
  ,('get bootstrap.tar.Z (replace')
  ,('get bootstrap.sh (replace')
  with nc
;

CL:OVRDBF FILE(INPUT) TOFILE(QTEMP/FTPCMD) MBR(*FIRST) OVRSCOPE(*JOB);
CL:OVRDBF FILE(OUTPUT) TOFILE(QTEMP/FTPLOG) MBR(*FIRST) OVRSCOPE(*JOB);

CL:FTP RMTSYS('public.dhe.ibm.com');

CL:QSH CMD('touch -C 819 /tmp/bootstrap.log; /QOpenSys/usr/bin/ksh /tmp/bootstrap.sh > /tmp/bootstrap.log 2>&1');

select
case when (message_tokens = X'00000000')
```

RPM bootstrap

```
create or replace table qtemp.ftpcmd(cmd char(240)) on replace delete rows;
create or replace table qtemp.ftplog(line char(240)) on replace delete rows;

insert into qtemp.ftpcmd(CMD) values
   ('anonymous anonymous@example.com')
  ,('namefmt 1')
  ,('lcd /tmp')
  ,('cd /software/ibmi/products/pase/rpms')
  ,('bin')
  ,('get README.md (replace')
 ,('get bootstrap.tar.Z (replace’)
 ,('get bootstrap.sh (replace')
 with nc
;
```

RPM bootstrap

```
CL:OVRDBF FILE(INPUT) TOFILE(QTEMP/FTPCMD) MBR(*FIRST) OVRSCOPE(*JOB);
CL:OVRDBF FILE(OUTPUT) TOFILE(QTEMP/FTPLOG) MBR(*FIRST) OVRSCOPE(*JOB);

CL:FTP RMTSYS('public.dhe.ibm.com');

CL:QSH CMD('touch -C 819 /tmp/bootstrap.log; /QOpenSys/usr/bin/ksh /tmp/bootstrap.sh > /tmp/bootstrap.log 2>&1');

select
case when (message_tokens = X'00000000')
 then 'Bootstrapping successful! Review /tmp/README.md for more info'
 else 'Bootstrapping failed. Consult /tmp/bootstrap.log for more info'
end as result
from table(qsys2.joblog_info('*')) x
where message_id = 'QSH0005'
order by message_timestamp desc
fetch first 1 rows only;
```

Installing RPM/YUM support

- **Step 3**: Once the installation of the bootstrap is complete, start a terminal session
  - This can be done via 5250 command `'call qcmd'`
  - Better yet, an SSH session can be established to the system
- **Step 4**: Modify the PATH to include the bin directory for the packages installed by the bootstrap
  - `PATH=/QOpenSys/pkgs/bin:$PATH  export PATH`

Useful commands

Command	Decscription
bash	A shell typically available on Linux systems. Features include command/file completion, and command recall.
gcc	GNU c Compiler
rpm	Used to install/manage packages built using the Redhat Package Manager.
yum	Yellowdog Updated, Modified – a wrapper around RPM that uses package repositories to simplify package installation and dependency resolution

Configuration de l’environnement utilisateur dans PASE

Création de l’environnement utilisateur

- Un certain nombre d’étapes doivent être accomplies pour créer l’environnement utilisateur
- Step 1: Create the user's home directory
  - `mkdir /home/<username>`
- Step 2: Create a .profile in the user's home directory. The .profile is used to define the shell environment, including environment variables, scripts to execute, and other commands. The `.profile` is used to store pre-defined settings when a shell program starts

```
PATH=/QOpenSys/pkgs/bin:$PATH
export PATH
bash
```

  - The first two lines update the path statement to include the location of the programs installed both by the bootstrap as well as subsequent `'yum install'` commands
  - The third line causes the bash shell to be executed
    - NOTE: by default a PASE terminal session starts the 'ksh' shell

Modification du shell par défaut

- Le shell par défaut dans l'environnement PASE est ksh (un favori dans l'espace AIX)
- Une meilleure alternative à ksh est bash - un favori dans l'espace Linux, en particulier pour ses fonctionnalités telles que le rappel de commandes (flèche vers le haut) et la complétion de nom de fichier (onglet).

- La fonction '`qsys2.set_pase_shell_info`' permet de changer le shell par défaut par utilisateur ou pour tous:

```
call qsys2.set_pase_shell_info('*DEFAULT',  '/QOpenSys/pkgs/bin/bash');
```

- Contrôle du default shell:

```
select authorization_name, pase_shell_path  from qsys2.user_info
 where pase_shell_path is not null;
```

Repository definition

- The RPM packages reside in a repository that is publicly accessible
- The definition of the repository is located in the
- `/QOpenSys/etc/yum/repos.d` directory
  - The repository file for the IBM RPM pile is ibm.repo

- `[ibm] name=ibm`
- `baseurl=http://public.dhe.ibm.com/software/ibmi/products/pase/rpms/repo enabled=1`
- `gpgcheck=0`

Note: it is possible to use a local repository by downloading the files from the indicated FTP site and then uploading them to a directory on the system. The `'baseurl'` would change to indicate `'file'` and the path to the directory of RPMs.

Additional note: ACS has support for cloning the repository to a local server

Evironment variables

- An environment variable is a name/value pair that can affect running process within a computing environment
- The current environment can be output with the 'env' command

```
$
> env
LANG=EN_BE
QIBM_USE_DESCRIPTOR_STDIO=I
TERM=xterm
TRACEOPT=UNLINK
QIBM_DESCRIPTOR_STDERR=CRLN=N
QIBM_DESCRIPTOR_STDOUT=CRLN=N
QIBM_DESCRIPTOR_STDIN=CRLN=Y
LOGNAME=KOEN
SHLVL=1
HOSTTYPE=powerpc
HOSTID=192.168.2.40
HOSTNAME=S7824EF0.CDINVEST.BE
OSTYPE=os400
MACHTYPE=powerpc-ibm-os400
TERMINAL_TYPE=5250
```

```
HOME=/home/KOEN
PATH=/QOpenSys/usr/bin:/usr/ccs/bin:/QOpenSys/usr/bin/X11:/usr/sbin:.:/usr/bin
PASE_PATH=/QOpenSys/usr/bin:/usr/ccs/bin:/QOpenSys/usr/bin/X11:/usr/sbin:.:/usr/bin
PASE_LANG=EN_BE
QIBM_PASE_CCSID=1208
PASE_LOCPATH=/usr/lib/nls/loc
LOCPATH=/usr/lib/nls/loc
PASE_NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat:/usr/lib/nls/msg/EN_BE/%N:/usr/lib/nls/msg/EN_BE/%N.cat
PASE_LC__FASTMSG=true
LC__FASTMSG=true
PASE_TZ=<CET>-1<CEST>,M3.5.0,M10.5.0
TZ=<CET>-1<CEST>,M3.5.0,M10.5.0
QIBM_IFS_OPEN_MAX=66000
```

Evironment variables – most important

Variable	Description
USER	Current user logged into system and using the current shell
PWD	The current working directory. This is the 'focus' of any command run on the system
HOME	The user's home directory. This is the directory that is typically used for storage of configuration files that affect a user's login environment as well as shell characteristics. NOTE: This directory does not exist by default in the IBM i environment
SHELL	The current shell. The environment supports multiple shells including bash and ksh
LOGNAME	The login name of the user
OLDPWD	The previous (n-1) working directory

Qu’est-ce qu’un shell ?

Qu’est-ce qu’un shell

- The command line used on “Unix™” systems (as well as Unix-like systems) as well as PASE
- Like CL it can be used interactively, or run as a program
- Like CL most commands are actually programs that get called
  - There are some “built in” commands
- Unlike CL there are a number of varieties of shell
    - - sh = bourne shell
    - - csh = C shell
    - - ksh = korn shell
    - - bash = bourne again shell
    - - qsh = Q shell
- There are some similarities but also differences
- Most of the discussion here is <u>not</u> operating system specific

  Will work on AIX, Linux, QSH in OS/400, other nasty Unix variants, etc

Why do we care about the shell

- All system configuration operations can be done through the shell – often more quickly then through a GUI

- Shell scripts can automate routine tasks such as backups, scheduled emails, etc.

- GUI can be used for a great amount of admin activities

  However, the shell tends to be a comfort zone providing ability to fix things in case something goes wrong

Different types of shell

- A number of shells are available each providing function/usability customized to a particular type of user:

- Popular shells include:
  - BASH (Bourne Again Shell)
  - PDKsh (Public Domain Korn Shell)
  - csh (C shell)
  - mc (Midnight Commander)
  - QSHELL (PASE shell)
  - ksh (Korn shell, default on AIX)

- Difference tends to be in scripting capabilities and user interface
  - Items such as command recall and file name completion are typically different

Starting with bash

Available in PASE!

- Bash stands for Bourne Again Shell
    - Started by Brian Fox in 1987
    - One of the most popular shells available on Linux

- Bash incorporates features of the Korn and C shell (`ksh` and `csh`)
- Bash configuration files:

/bin/bash	Bash executable
/etc/profile	System wide initialization file for login shells
~/.bash_profile	Personal initialization file for login shells
~/.bashrc	Personal per-interactive-shell startup file
~/.bash_logout	Login shell clean file that executes when shell exits

A little more on bash

- Default Linux shell
  - This can be changed in a variety of ways
    - `/etc/profile` – login shell
    - `$HOME/.profile`
- As we saw earlier, can be set as the default shell for PASE
- Very powerful as a command line shell
  - Recall previous commands
  - Command and file completion with the <TAB> key
- Many programming features
  - Loops and conditionals

Shell environment

The shell is an environment where commands can be entered and the Operations system can respond to them

A key concept to the environment is environment variables

- There are a large number of environment variables
- **HISTFILE**: points to file containing the shell history, defaults to `~/.bash_history`
- **HISTFILESIZE**: how man last commands you wish to have in history
- **HOME**: points to your home directory
- **PATH**: set of directories to search when trying to execute a command
- **PS1**: Prompt variable
- **USER**: username

NOTE: All of these environment variables are available when running bash in the PASE environment.

Exploring the shell

- The shell is the command-interpreter and as such there are a number of features that make it easier to work in and traverse the environment

- The shell keeps a history of previous commands that have been executed. The 'history' command can be used to display a list of those commands:
  - `-bash-4.4# history`
    `1 pwd`
    `2 ls -l`

NOTES:

The commands are shown preceded with a number. Any command in the history can be re-executed simply by entering !
Followed by the number from the command history list

Previously executed command can be recalled through use of the up-arrow key.
This allows you to scroll through previously executed commands.

Exploring the shell

- A powerful feature of the bash shell is file-name completion
- File-name completion is accomplished by pressing the <TAB> key after entering a portion of a file-name.
  - At this point the shell will complete as much of the name as possible while remaining unique
  - If there are multiple names that match what has been entered then pressing the <TAB> key twice will show those matches.

    - `type ls /QO  Press <Tab>`
    - Notice that the shell completes /QOpenSys/
    - `Press <TAB> twice`
    - Notice that the shell provides a list of items under /QOpenSys/

Shell metacharacters

Symbol	Description
>	Output redirection
>>	Output redirection (append)
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; one character
[]	File substitution wildcard; any character between brackets
`cmd`	Command substitution
$(cmd)	Command substitution
|	The pipe (connect output of command on right to input on command on left)
;	Command sequence
||	OR conditional execution
&&	AND conditional execution

Shell metacharacters

Symbol&	Description
()	Group commands
&	Run command in the background
#	Comment
$	Expand the value of a variable
\	Prevent escape interpretation of the next character
<<	Input redirection
“$val”	Literal with variable substitution
‘$val’	Literal without variable substitution

Why are metacharacters important

- The shell has two primary responsibilities
  - Walk the command-line looking for tokens
  - Cause a command string to be sent to the kernel

- Tokens are identified by white-space

- The metacharacters are considered tokens

- `ls -l /home/koen > /tmp/listing`

Usefull shell constructs

- **Arrow Up & Down**: Scroll through recent commands used

- **&&:** command is only executed if preceding command was successful:
  - `command1 && command2`

- **alias**: sets a command alias or prints defined aliases
  - `alias wrklnk=ls`

- **bg[jobid]**: Resumes the suspended job in the background

- **cd**: changes current directory to directory indicated
  - `cd /home`

Usefull shell constructs

- **echo**: Outputs the arguments
  - `echo "hello world"`
- **find [path][expression]**: searches the directory indicating looking for files that match expression:
  - `find / -name passwd -print`
- **pwd** – Prints the absolute pathname of the current working directory
- **unalias** – Removes an alias
- **history** – displays command history with line numbers
- **umask** – is a command that determines the settings of mask to control how file permissions are set for new files
- **logout**: exits the shell environment
- **exit [n]:** exits shell environment with exit status n

‘bash’-ing PASE

- The bash shell is available for PASE
  - It is part of the RPM pile

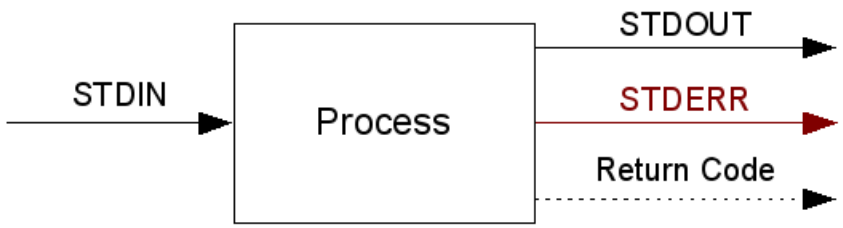
- Step 1: Install the RPM pile bootstrap

- Step 2: Install bash
  - `yum install bash`

Input and Output

nix programs start with three open files

- Input (called `stdin`) (#0)
- Output (called `stdout`) (#1)
- Error output (called `stderr`) (#2)



We can redirect the output to go to a file by using ">"

- `ls -l > output.txt`

This will take the output of the "`ls -l`" command and write it into a file called "`output.txt`"

Input and Output

You can also specify *which* output goes to a file

- `ls -l 1> output.txt`

this is the same as before.

Redirecting only error output

- `grep fred * 2> grep.err`

this will redirect only the error output to the file `grep.err`

`Note : 1 = stdout, 2 = stderr`

Input and Output

Use the "<" operator to redirect input

Equivalent to typing at the keyboard

For example

- `sed "s/koen/Koen/g" < my.txt`

This runs the command "`sed`" (an editor) changing “koen" to “Koen" and taking its input from a file called "`my.txt`"

Input and output

- `sed "s/koen/Koen/g" < my.txt > my2.txt`

Common *nix commands

Unix is built around the idea of lots of little programs that all do one thing well. Shell programs generally involve stringing lots of these together

ls	lists files
sed	an editor
grep	a searcher
cat	a file outputer
find	a file locator
sort	sorts files
tr	translates characters
ps	list processes
seq	print a sequence of numbers
…	a thousand others

All of these are available in the IFS and can be run from an IBM i shell (i.e., qsh, ssh session)

Many available on PASE and more coming!

Help for all these is in the "man" (for "manual") command => man sed Unfortunately man is not available in PASE by default, you need to install the packages

Pipes

Standard output (STDOUT) of one command/program is used as the standard input (STDIN) for the next command/program

- *STDOUT* (1) → *STDIN* (0)
  - `ps -x | grep java`

    List processes, search for any involving java

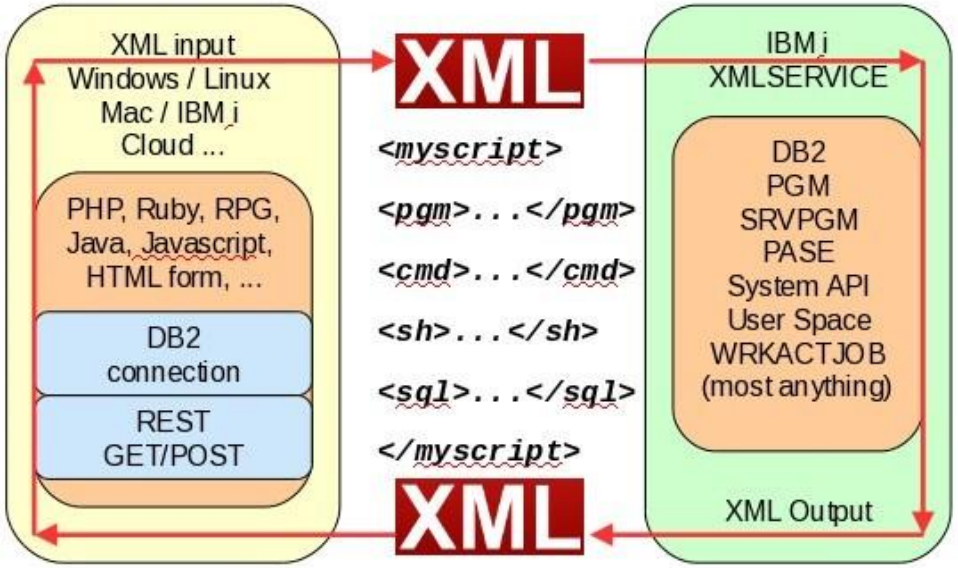
Only the 'STDOUT' from the last command is actually output to the screen (unless re-directed) NOTE: Any output to STDERR by the commands will be output to the screen – again unless re-directed.

NOTE: pipes are not limited to two commands… any number of pipes can be used to build a pipeline: `cmd1 | cmd2 | cmd3 | …. | cmdx`

XMLSERVICE

XMLService Overview

- XMLService provides the ability to invoke programs in the ILE (RPG, Cobol) as well as CL commands
- ILE items such as data areas and data queues can also be accessed
- XMLService uses XML payloads both to define the request and provide the results
- XMLServicecan be invoked from any language



http://yips.idevcloud.com/wiki/index.php/XMLService/XMLService

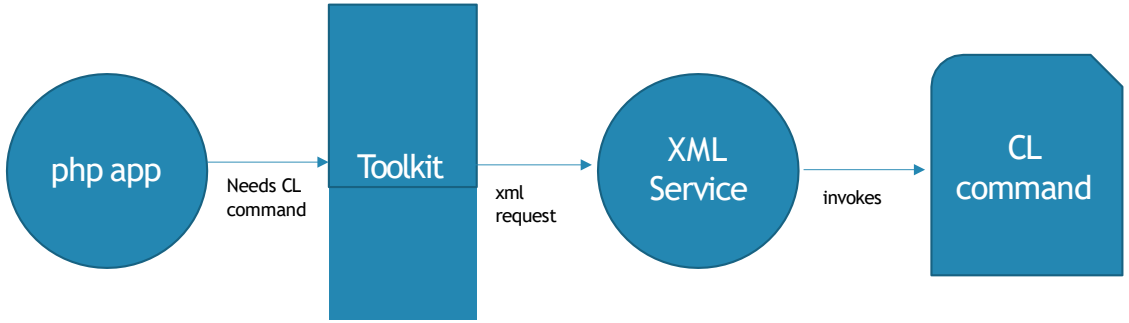
The Toolkit

- While it is possible to build the XML payload "manually", most open-source languages provide a language-specific toolkit to make building of the XML payload as well as dehydrating of the results easier.

- The Toolkit is a set of classes that essentially "wrap" the calls to the XMLService.

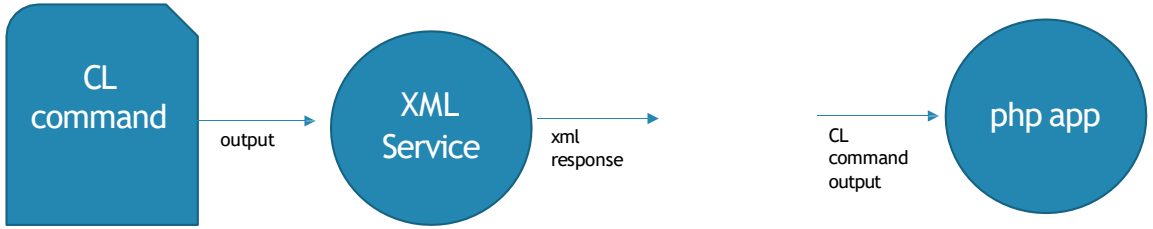
- The toolkit is Object-Oriented based; however, the OO is easy to understand.

CL command invocation – Call Flow [request]



- The toolkit is used to encapsulate the request to XML.
- The toolkit provides the XML request to the XMLService via the HTTP transport mechanism
- XMLService dehydrates the XML and invokes the requested CL command

CL command invocation – Call Flow [response]



- Output from the CL command is provided to the XMLService
- XMLService packages the output in an XML-formatted response
- Response is provided to the Toolkit
- Toolkit dehydrates the XML and provides the CL command output to the PHP application.

XMLService is not intended to replace stored procedures
Stored procedures typically provide better performance than XMLService
For environments that have stored procedures and a comfort level using them can continue to be used.

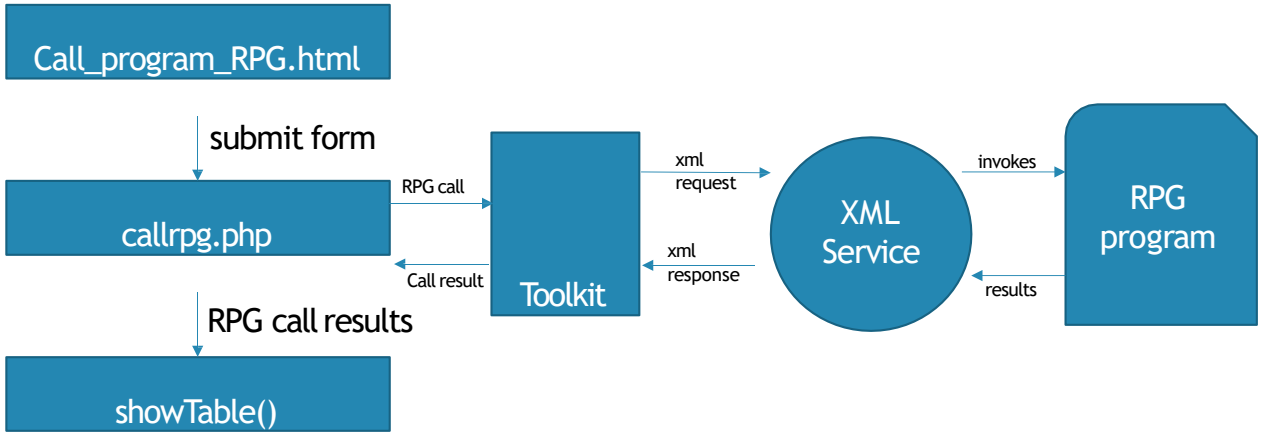
Peering under the Covers – What does the XML Look Like

The call

The result
Each CDATA represents a line of output from command execution

```
1
2  Exec start: 2020-05-08 14:37:15
3  Version of toolkit front end: 1.7.0
4  IPC: ''. Control key: *cdata *here
5  Stmt: call zendphp7.iPLUG32K(?,?,?,?) with transport: ibm_db2
6  Input XML: <?xml version="1.0" encoding="ISO-8859-1" ?>
7  <script>
8  <sh rows='on'>/QOpenSys/usr/bin/system "DSPLIBL"</sh>
9  </script>
10 Output XML: <?xml version="1.0" encoding="ISO-8859-1" ?>
11 <script>
12 <sh rows='on'>
13 <row><![CDATA[ 5770SS1 V7R3M0  160422                    Library List
14 <row><![CDATA[                          ASP]]></row>
15 <row><![CDATA[   Library     Type       Device      Text Description]]></row>
16 <row><![CDATA[   QSYS        SYS                    System Library]]></row>
17 <row><![CDATA[   QSYS2       SYS                    System Library for CPI's]]></row>
18 <row><![CDATA[   QHLPSYS     SYS]]></row>
19 <row><![CDATA[   QUSRSYS     SYS                    System Library for Users]]></row>
20 <row><![CDATA[   QGPL        USR                    General Purpose Library]]></row>
21 <row><![CDATA[   QTEMP       USR]]></row>
22 <row><![CDATA[                          * * * * *  E N D  O F  L I S T I N G  * * * * *]]></row>
23 </sh>
24 </script>
25 Exec end: 2020-05-08 14:37:15. Seconds to execute: 0.12581586837769.
```

RPG Program Call – Architectural View



Node.js calling ILE

It's All Open – It's All Available

https://github.com/IBM / nodejs-itoolkit

IBM / nodejs-itoolkit

Watch 13 | Star 4 | Fork 6

Code | Issues 15 | Pull requests 0 | Projects 1 | Insights

Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

A JavaScript (Node.js) library for communicating with IBM i

ibmi | xmlservice | nodejs | nodejs-modules

37 commits | 2 branches | 3 releases | 7 contributors | MIT

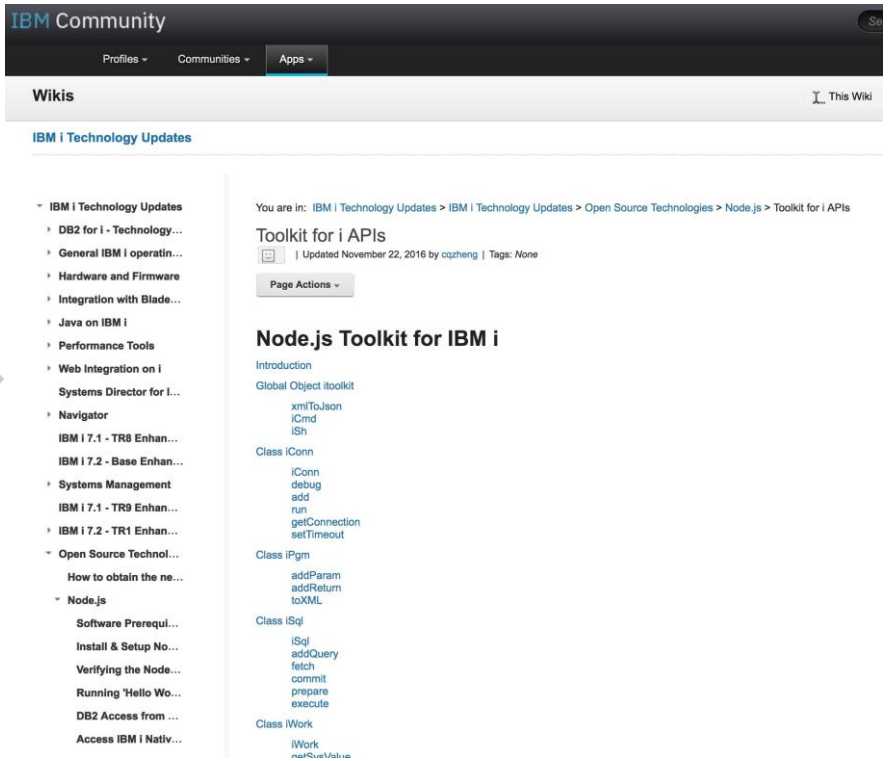
Branch: master | New pull request | Find File | Clone or download

abmusse Merge pull request #43 from jkdavew/passByValue ... Latest commit 98b8b3e 21 days ago

.github	Add contribution guidelines (#37)	a month ago
lib	- ensure <param> tag contains the 'by' attribute for val/ref.	23 days ago
test	- consolidated test into iPgmUnit	23 days ago
.gitignore	added some missing default values	3 months ago
.npmignore	Add contribution guidelines (#37)	a month ago
CONTRIBUTING.md	Add contribution guidelines (#37)	a month ago
LICENSE	Initial commit	2 years ago
README.md	Add contribution guidelines (#37)	a month ago
contributors.txt	Initial commit	2 years ago
package.json	Update package.json	2 months ago

Toolkit API Reference

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20i%20Technology%20Updates/page/Toolkit%20for%20i%20APIs?section=iSh



Installing the Toolkit

- The installation is performed from a PASE shell

```
$ npm i itoolkit
```

```
# npm i itoolkit
npm WARN saveError ENOENT: no such file or directory, open
'/QOpenSys/pkgs/lib/nodejs8/include/node/package.json'
npm notice created a lockfile as package-lock.json. You should commit this
file.
npm WARN enoent ENOENT: no such file or directory, open
'/QOpenSys/pkgs/lib/nodejs8/include/node/package.json'
npm WARN node No description
npm WARN node No repository field.
npm WARN node No README data
npm WARN node No license field.


+ itoolkit@0.1.6
added 1 package from 4 contributors and audited 1 package in 1.508s
found 0 vulnerabilities
```

Basic Toolkit Flow

Establish a connection to XMLSERVICE

```
var xt = require("itoolkit");
var conn = new xt.iConn("*LOCAL", "USERNAME", "PASSWORD");
```

Define function for XML to JSON conversion

```
function cbJson(str) {
  var result = xt.xmlToJson(str);
  console.log(JSON.stringify(result, " ",2))
 }
```

Build the command / shell script / program call / SQL call

```
conn.add(xt.iCmd(…
conn.add(xt.iSh(…
var Pgm = new xt.iPGM….    / conn.add(pgm);
var sql = new xt.iSQL….   / conn.add(sql);
```

Run the command list

```
conn.run(cbJsdon);
```

Toolkit Example – Basic APIs Initial Include and Connection

```
var xt = require("itoolkit");
var conn = new xt.iConn("*LOCAL", "USERNAME", "PASSWORD");
```

- The '`require`' statement causes the class definition for the toolkit to be included
  - xt is an object that represents the methods (functions) and properties (variables) of the `itoolkit` class

- The '`var conn`' statement causes the `iConn` method to be invoked
  - This establishes a connection to the XMLSERVICE
  - The variable '`conn`' represents the connection

Basic APIs Converting XML to JSON

```
function cbJson(str) {
  var result = xt.xmlToJson(str);
  console.log(JSON.stringify(result, " ", 2))
}
```

- The `xmlToJson` converts the output XML document into JSON format.
  - XMLService returns an XML document
  - JSON is more compatible with the Java script language

- The function will be passed as a parameter to the `run` method from the toolkit object which causes the command list to be executed.

Basic APIs Calling CL and QSHELL Commands

```
conn.add(xt.iCmd("RTVJOBA USRLIBL(?) SYSLIBL(?)"));
```

- This statement adds the `'RTVJOBA'` command to the command list.
- Multiple commands can be put on the command list prior to execution.

```
conn.add(xt.iSh("system -i wrksyssts"));
```

- This statement adds the `'system –i wrksyssts'` command as a QSHELL command to the command list.

Basic APIs Program/Service Call

Build a program call with all necessary I/O parameters and add it to the command list

```
var pgm = new xt.iPgm("QWCRSVAL", {"lib":"QSYS"});
var outBuf = [
               [0, "10i0"],
               [0, "10i0"],
               ["", "36h"],
               ["", "10A"],
               ["", "1A"],
               ["", "1A"],
               [0, "10i0"],
               [0, "10i0"]
        ];
pgm.addParam(outBuf, {"io":"out"});
pgm.addParam(66, "10i0");
pgm.addParam(1, "10i0");
pgm.addParam("QCCSID", "10A");
pgm.addParam(this.errno, {"io":"both", "len" : "rec2"});
conn.add(pgm);
```

Basic APIs SQL Statement

Build an SQL call and add it to the command list

```
var sql = new xt.iSql();
sql.prepare("call qsys2.tcpip_info()");
sql.execute();
sql.fetch();
sql.free();
conn.add(sql);
```

Toolkit Example – Basic APIs
Execute the Command List

```
conn.run(cbJson);
```

Note the `cbJson` parameter which is the function to convert the XML (returned by XMLService) to JSON

Toolkit Capabilities

The toolkit capabilities are based on various classes provided by the toolkit

Class	Description
iConn	Provides various methods for establishing and workingwith connections between Node.js and IBM i
iPgm	Provides methods for working with programs andservice programs
iSql	Provides methods for working with SQLstatements
iWork	Provides methods for working with system values andstatus information as well as retrieving data areacontents.
iProd	Provides methods for working with productinformation.
iUserSpace	Provides methods for working with UserSpaces
iNetwork	Provides methods for working with NetworkInformation
iObj	Provides methods for working withObjects
iDataQueue	Provides methods for working with DataQueues

PHP calling ILE

Toolkit is Object Oriented

- Series of classes that "wrap" the IBM project
- All PHP but Object Oriented...(wait, there's more)
- The Toolkit is Open Source
- No OO training required to use them!!!
- Nothing like OPO!

Toolkit Service Class Documentation: http://files.zend.com/help/Zend-Server/content/toolkit_service_class.htm

Simple CL Command Execution

```
<?php
require_once zend_deployment_library_path('PHP Toolkit for IBMI i') .
'/ToolkitService.php';
try {
    $obj = ToolkitService::getInstance();
} catch (Exception $e) {
    exit($e->getMessage());
}
$obj->setOptions(array(
        'stateless' => true,
        'debug'     => true,
        'debugLogFile' =>_DIR . '/toolkit-debug.log',
        'plugSize'  => '32K'
    )
);

if (! ($rows = $obj->CLInteractiveCommand('DSPLIBL'))) {
    echo $obj->getLastError();
} else {
    echo '<pre>' . print_r($rows, true) . '</pre>';
}
$obj->disconnect();
```

Include ToolkitService class

Establish an instance of the Toolkit class

Establish parameters for upcoming call

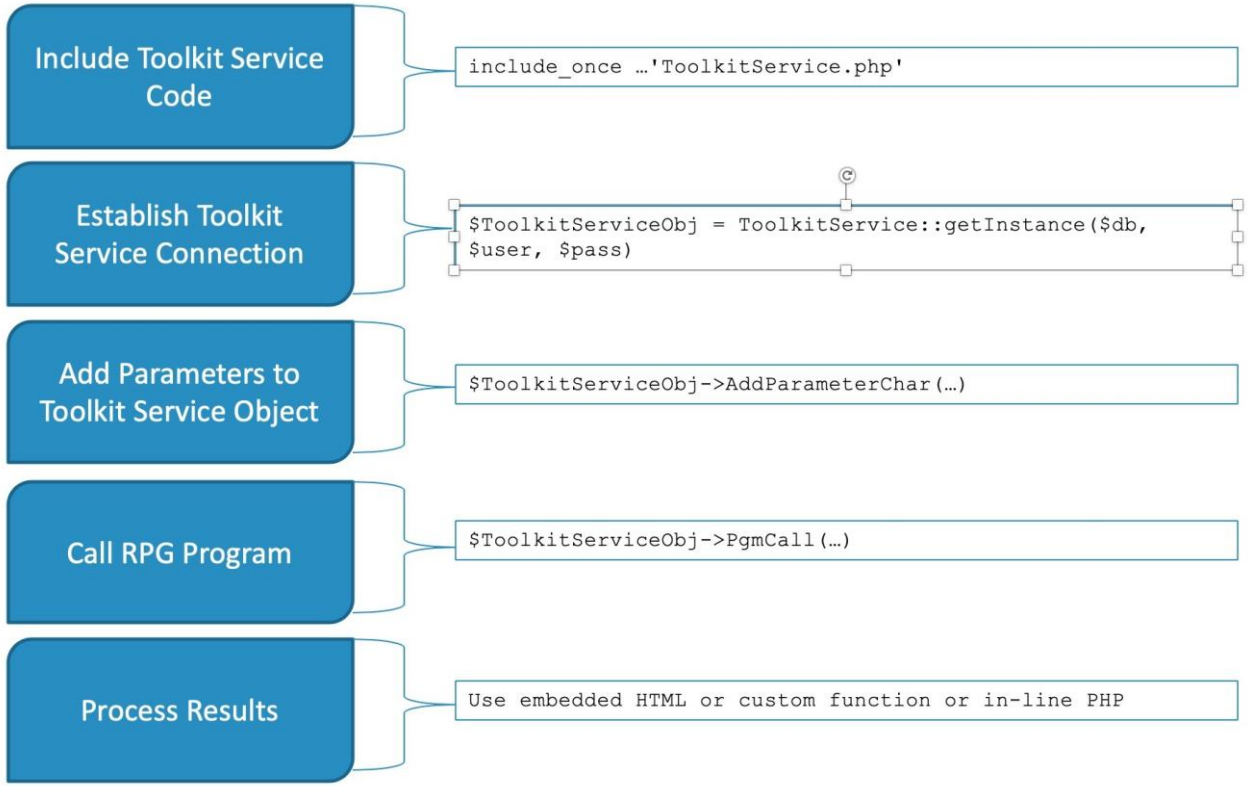
Call CLIneractiveCommand method to invoke the DSPLIBL CL command

Terminate the toolkit connection

Simple CL Command Execution - Output

```
Array
(
    [0] =>  5770SS1 V7R3M0  160422                Library List                                  5/09/20 10:51:14       Page    1
    [1] =>                          ASP
    [2] =>    Library     Type      Device      Text Description
    [3] =>    QSYS        SYS                   System Library
    [4] =>    QSYS2       SYS                   System Library for CPI's
    [5] =>    QHLPSYS     SYS
    [6] =>    QUSRSYS     SYS                   System Library for Users
    [7] =>    QGPL        USR                   General Purpose Library
    [8] =>    QTEMP       USR
    [9] =>                          * * * * *  E N D  O F  L I S T I N G  * * * * *
)
```

Invoking RPG



The RPG Program Being Called

```
0100    *****************************************************************
0300    C     *ENTRY        PLIST
0400    C                   PARM                    CODE            10
0500    C                   PARM                    NAME            10
0600    *****************************************************************
0700    C     CODE          IFEQ      '1'
0800    C                   movel     'IBM'         name
0900    C                   ELSE
1000    C     CODE          IFEQ      '2'
1100    C                   movel     'Zend'        name
1200    C                   ELSE
1300    C                   movel     'wrong code'  name
1400    C                   ENDIF
1500    C                   ENDIF
1600    C*
1700    C                   SETON                                       LR
```

It is important to understand the parameters, their type and size as well as the usage (input, output, or both) when building up the call from the PHP program.
When developing PHP code to call RPG programs it is recommended that a single function be developed for each RPG program that will be called.

- The program being called has the following attributes
  - Parameter-name CODE length 10 – this value is being passed into the RPG program
  - Parameter-name NAME length 10 – this value is being returned from the RPG program
- The program name is 'COMMONPGM'
- The program resides in the ZENDPHP7 library
- The program logic indicates that the input parameter (CODE) will be tested
  - A value of '1' returns 'IBM' in the NAME parameter
  - A value of '2' returns 'ZEND' in the NAME parameter
  - Any other input value returns 'WRONG CODE' in the name parameter.

The PHP Script – Part 1

```
<?php
require_once zend_deployment_library_path('PHP Toolkit for IBMI i') .
'/ToolkitService.php';
use ToolkitApi\Toolkit;
try {
    $obj = ToolkitService::getInstance();
} catch (Exception $e) {
    exit($e->getMessage());
}
$obj->setOptions(array(
        'stateless' => true,
        'debug' => true,
        'debugLogFile' => '/tmp/toolkit-debug.log',
        'plugSize' => '32K'
    )
);
```

- Same as the CL example
  - Include the class definition for the toolkit
  - Set options

The PHP Script – Part 2

```
$param = array();
$code = isset($_POST['code']) ? $_POST['code'] : ' ';
$desc = ' ';
$param[] = Toolkit::AddParameterChar('both', 10, 'Input Code', 'CODE', $code);
$param[] = Toolkit::AddParameterChar('both', 10, 'Output Desc', 'DESC', $desc);
$result = $obj->pgmCall('COMMONPGM', 'ZENDPHP7', $param);
$obj->disconnect();
if (! $result) {
    exit('Execution failed');
}
?>
```

- Add descriptions/attributes for the parameters expected by the RPG program to the $param array. (described in more detail on subsequent slide)
- Invoke PgmCall() to request execution of the program
- Disconnect from XMLService
- Test for valid result

The PHP Script – Part 3

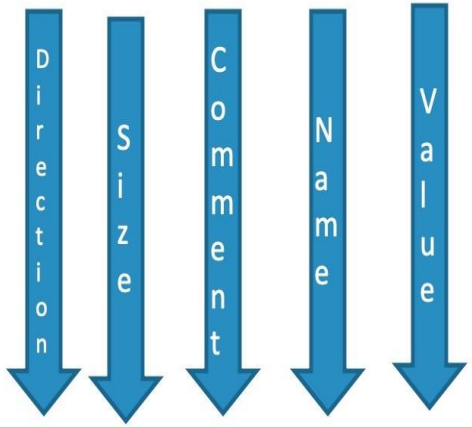
```
<!DOCTYPE html>
<html lang="en">
<head><style>table, th, td { border: 1px solid black}</style></head>
<body>
<table>
    <tr><th>Parameter Name</th><th>Parameter Value</th></tr>
    <?php foreach($result['io_param'] as $key => $value) { ?>
        <tr><td><?= $key ?></td><td><?= $value ?></td></tr>
    <?php } ?>
</table>
<a href="Call_program_RPG.html">Return to Input Form</a>
</body>
```

- HTML with embedded PHP to output the results of calling the RPG program.

The RPG Program Call – Defining the Parameters

Parameters to the call:

- Data Direction Type
- Parameter Size
- Developer Comment
- Parameter Name
- Parameter Value




```
$param[] = $ToolkitServiceObj->AddParameterChar('both', 10, 'CODE', 'CODE', $code);
$param[] = $toolkitServiceObj->AddParameterChar('both', 10, 'DESC', 'DESC', $desc);
```

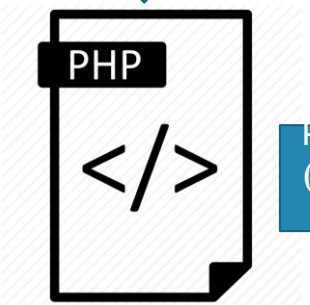
The Call and Result

Calling an RPG program with two parameters

Code: 1 **(Valid Code is 1 or 2)**

Reset Call program

HTML form invokes PHP script



PHP script calls RPG program (via XMLService) and outputs result in HTML response

Parmeter Name	Parameter Value
CODE	1
DESC	IBM

Service Program Call – Page 1

```
<?php

require_once zend_deployment_library_path('PHP Toolkit for IBMI i') .
'/ToolkitService.php';
try {
    $obj = ToolkitService::getInstance();
} catch (Exception $e) {
    exit($e->getMessage());
}
$obj->setOptions(array(
        'stateless' => true,
        'debug' => true,
        'debugLogFile' => '/tmp/toolkit-debug.log',
        'plugSize' => '32K'
    )
);
```

- Same as the CL and RPG examples:
  - Include the class definition for the toolkit
  - Set options

Service Program Call – Page 2

```
$sysValueName = 'QCCSID';
$error = ' ';
$value= ' ';
$param = array();
$param[] = $obj->AddParameterChar('both', 1,'ErrorCode','errcode',  $error);
$param[] = $obj->AddParameterChar('both', 10,'SysValName','sysvalname', $sysValueName);
$param[] = $obj->AddParameterChar('both', 1024,'value','sysvalue', $value);
$result = $obj->PgmCall('ZSXMLSRV', "ZENDPHP7", $param, null, array('func'=>'RTVSYSVAL') );
if (! $result) {
    $obj->disconnect();
    exit("Operation failed. System value $sysValueName not retrieved. Exiting...");
}
print "System value $sysValueName = {$result['io_param']['sysvalue']}";
```

- Build up the parameter list. Make note of the $sysValueName being set to QCCSID which is going to result in the QCCSID being returned
- Use the PgmCall() method to invoke the program call (ZSXMLSRV) and library ZENDPHP7 to invoke the RTVSYSVAL command with the defined parameters
- Test for a valid result
- Output the returned value (indexing into the result array based on the associative index of 'sysvalue')

Service Program Call – Page 3

```
/* change sysvalname parameter value from QCCSID to QLANGID and run PgmCall() again
PHP arrays are zero-based indexed. We're changing the $param[] on line 27 above . */
$sysValueName = 'QLANGID';
$param[1]->setParamValue($sysValueName);
$result = $obj->PgmCall('ZSXMLSRV', "ZENDPHP7", $param, NULL, array('func'=>'RTVSYSVAL') );
if (! $result) {
    $obj->disconnect();
    exit("Operation failed. System value $sysValueName not retrieved. Exiting...");
}
print " System value $sysValueName = {$result['io_param']['sysvalue']}";
$obj->disconnect();
```

- Change the second parameter of the call to 'QLANGID' which is going to result in retrieving the mnemonic variant of the QCCSID
- Execute the program call
- Output the returned value

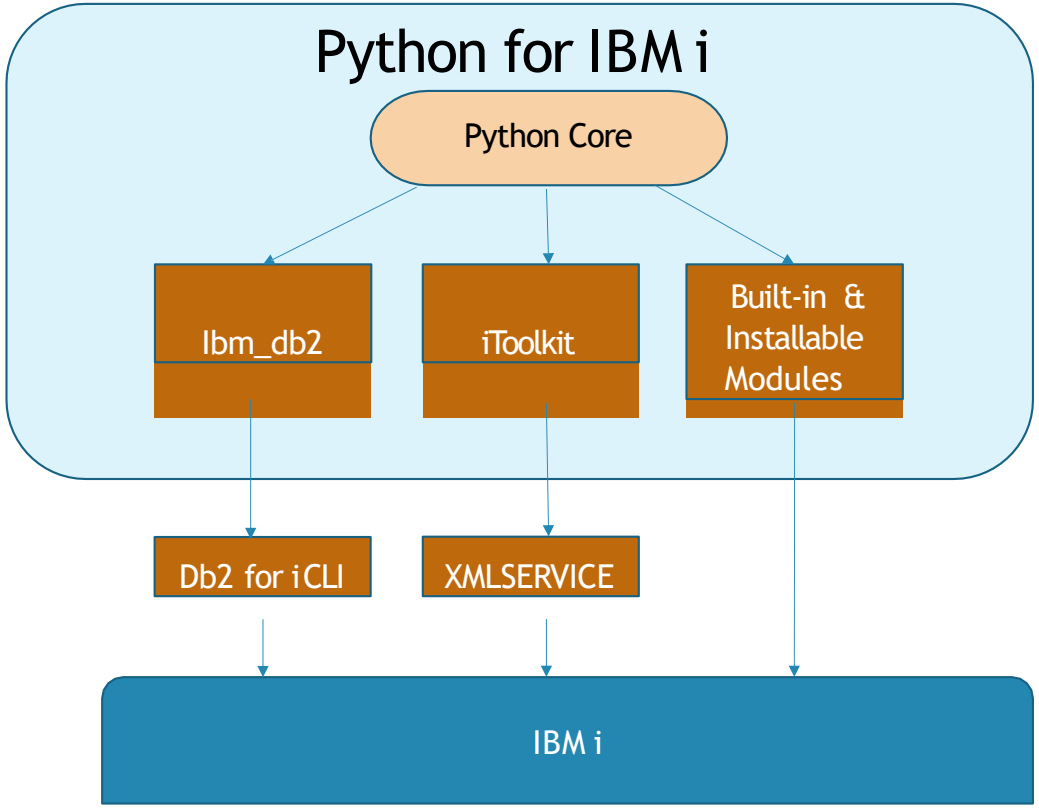
Service Program Call – Result / Output

System value QCCSID = 37 System value QCCSID = ENU

- The value of 37 is output as a result of the first PgmCall() where the SysValName parameter was set to QCCSID
- The value of ENU is output as a result of the second PgmCall() where the SysValName parameter was set to QLANGID

Python calling ILE

Python for IBM i Infrastructure



Overview

- itoolkit is a wrapper around XMLService that enables programs to call:
  - RPG programs and service programs
  - CL commands
  - PASE programs and shell scripts
  - SQL database access
- Useful Sites

Description	URL
Itoolkit project	https://ibm.biz/itoolkitpython
Python interface to XMLService	https://ibm.biz/xmlservice

What Can Be Done?

	iToolKit Command	Usage / Description
Commands	iCmd	Call CL command (even without output parameters)
	iCmd5250	Call CL command and get screen output
Programs	iPgm	Call Program
	iSrvPgm	Call Service Program
	iSh	Call PASE Program or ShellScript
	iXml	Anything else

Transports

Transport	Description	Notes

Database	Stored procedure call over database connection	• Any PEP-249 connection object can be used
		  • ibm_db_dbi
		  • pyodbc
		• No configuration needed using ibm_-db_dbi when used locally
		• Can use pyodbc with IBMI Access driver
		• Local and remote connections supported
		  • Using ibmdb_dbi remotely requires Db2 Connect License

HTTP	Calls using HTTP REST API	• Requires XMLSERVICE configured as a FastCGI endpoint inApache • Uses Apache TLS configuration forsecurity • Local and remote connections supported
Direct	Runs in current job (Fastest)	• No Configuration Needed • Somethings don't work from chroot (container) • Currently broken with 64-bit Python • Local connection only
SSH	Calls over SSH using Paramiko package	• Requires xmlservice-cli package installed on targetsystem • Uses SSH forsecurity • Local and remote connections supported

Connecting

```
# DirectTransport example

from itoolkit import *

from itoolkit.transport import DirectTransport


itransport = DirectTransport()


# HttpTransport example

from itoolkit import *

from itoolkit.transport import HttpTransport


itransport = HttpTransport(url, user, password)
```

Establish connection using Direct transport

Establish connection using HTTP transport

Connecting (Database Transport Example)

```
# DatabaseTransport examples
from itoolkit import *
from itoolkit.transport import DatabaseTransport

# using ibm_db
import ibm_db_dbi
itransport = DatabaseTransport(ibm_db_dbi.connect())

# using pyodbc
import pyodbc
itransport =
DatabaseTransport(pyodbc.connect("DSN=myDSN"))
```

Import itoolkit and Database Transport

Establish connection via ibm_db_dbi driver

Establish connection via pyodbc driver

Calling Standard CL Commands

```
itool.add(iCmd('addlible', 'ADDLIBLE TEST'))

itool.call(itransport)

rtvjoba = itool.dict_out('rtvjoba')

if 'error' in rtvjoba:
    print("Encountered an error")
```

Build the call to the ADDLIBLE command

Perform the call using the transport specified earlier

Get return from CL command call

Check for error

Output Parameters from CL Commands

```
itool.add(iCmd('rtvjoba', \
               'RTVJOBA USRLIBL(?) SYSLIBL(?)
CCSID(?N) OUTQ(?)'))
itool.call(itransport)
rtvjoba = itool.dict_out('rtvjoba')
if 'success' in rtvjoba:


    print(rtvjoba['row'][0]['USRLIBL'])
    print(rtvjoba['row'][1]['SYSLIBL'])
    print(rtvjoba['row'][2]['CCSID'])
    print(rtvjoba['row'][3]['OUTQ'])
```

Similar to previous example except uses parameters for call

If command execution succeeds, use array indexes to output specific values from return value

Output Parameters, Sensible

```
itool.add(iCmd('rtvjoba', \
               'RTVJOBA USRLIBL(?) SYSLIBL(?) CCSID(?N) OUTQ(?)'))
itool.call(itransport)
rtvjoba = itool.dict_out('rtvjoba')
if 'success' in rtvjoba:
    rtvjoba = { k: v for d in rtvjoba['row'] for k, v in d.items()  }
    print(rtvjoba['USRLIBL'])
    print(rtvjoba['SYSLIBL'])
    print(rtvjoba['CCSID'])
    print(rtvjoba['OUTQ'])
```

The highlighted line is the big difference from the previous example – essentially this statement builds up the 'rvtjoba' as an array which is indexed by key names returned from execution of the RTVJOBA command.

Command Display Output

Another example - this one using iCMD5250 to call a command and display it's output

```
itool.add(iCmd5250('wrkactjob_key', 'WRKACTJOB'))


itool.call(itransport)
wrkactjob = itool.dict_out('wrkactjob_key')
print(wrkactjob['wrkactjob_key'])
```

Command Display Output

```
Work with Active Jobs                                                    Page    1

5770SS1 V7R2M0 140418                                                                    DBCSB3P2 05/09/18 11:18:59 CDT

Reset . . . . . . . . . . . . . . . . :         *NO

Subsystems  . . . . . . . . . . . . . :         *ALL

CPU Percent Limit . . . . . . . . . . :         *NONE

Response Time Limit . . . . . . . . . :         *NONE

Sequence  . . . . . . . . . . . . . . :         *SBS

Job name  . . . . . . . . . . . . . . :         *ALL

CPU %  . . . :        .0          Elapsed time . . . . . .   00:00:00           Activ jobs . . . . . . :       233
. :                                                                              e
                                  Current                              --------Elapsed---------                                    Temporary

Subsystem/Job  User        Number   User        Type Pool  Pty     CPU   Int     Rsp  AuxIO   CPU%  Function       Status   Threads    Storage

QBATCH         QSYS        799117   QSYS        SBS    2   0        .4                  0     .0                  DEQW          2          3

  QDFTJOBD     JENKINS     846920   JENKINS     BCH    2  50        .0                  0     .0  CMD-QSH         EVTW          1          4

  QDFTJOBD     REDIS       846925   REDIS       BCH    2  50        .0                  0     .0  CMD-QSH         TIMW          1          4

  QZSHSH       JENKINS     846921   JENKINS     BCI    2  50     244.9                  0     .0  JVM-jenkins.wa  THDW         81        520

  QZSHSH       KADLER      846858   KADLER      BCI    2  50      14.6                  0     .0  PGM-python3     SELW          1         20

  QZSHSH       KADLER      849046   KADLER      BCI    2  50       1.0                  0     .0  PGM-python3     SELW          1         46
```

Calling an ILE program

Another example - this one using iPGM to call an RPG program

```
itool.add(iPgm('my_key','MYPGM')
    .addParm(iData('a', '1a', 'a'))
    .addParm(iDS('ds')
        .addData(iData('b', '10i0', '1'))
        .addData(iData('c', '12p2', '3.33'))))
itool.call(itransport)
results = itool.dict_out('my_key')
print(results['a'])
print(results['ds']['b'])
print(results['ds']['c'])
```

Use iPGM to specify the RPG program to call with necessary parameters

Perform the call using the transport specified earlier

Obtain and output the results

Calling a Service Program

Last example - this one using iSrvPGM to call a Server Program

```
itool.add(iSrvPgm('my_key','MYSRVPGM', 'myfunction')
    .addParm(iData('a', '1a', 'a'))
    .addParm(iDS('ds')
        .addData(iData('b', '10i0', '1'))
        .addData(iData('c', '12p2', '3.33'))))

itool.call(itransport)
results = itool.dict_out('my_key')
print(results['a'])
print(results['ds']['b'])
print(results['ds']['c'])
```

Invoke the 'myfunction' from the 'MYSRVPGM' service program

Specify parameters for the call

Perform the call using previously configured transport

Make the call, output the results

Toolkit review

Toolkits

- Toolkits provide the ability to integrate with various features/functions/programs of IBMI
- Node.s toolkit:
  - https://bitbucket.org/litmis/nodejs-itoolkit
- PHP Toolkit for IBM i:
  - http://yips.idevcloud.com/wiki/index.php/XMLSERVICE/Python
  - https://bitbucket.org/litmis/python-itoolkit
- Python itoolkit-lite
  - https://bitbucket.org/litmis/nodejs-itoolkit
- Ruby itoolkit
  - https://bitbucket.org/litmis/ruby-itoolkit
- Swift
  - https://bitbucket.org/litmis/swift-itoolkit
- .NET
  - https://github.com/richardschoen/IbmiXmlserviceStd

NOTE: Documentation for Ruby, Swift, and .NET is not sufficient to complete the capability information provided on the following slides

Toolkit Capabilities – Access Db2

	Node.js	PHP	Python
Execute direct SQL statements		Yes	Yes
Prepare SQL statements (place holders to prevent SQL injection)	Yes	Yes	Yes
Execute prepared statements	Yes	Yes	Yes
Fetch result sets	Yes	Yes	Yes
Return the number of fields in a result set	No	Yes	No
Retrieve columns and associated privileges for a table	No	Yes	No
Retrieve columns and associated metadata for a table	No	Yes	No
Fetch return sets with associated column headers	No	Yes	No
Retrieve field information	No	Yes	No
Retrieve primary keys for a table	No	Yes	No
Retrieve stored procedures registered in a database	No	Yes	No
Rollback a transaction	No	Yes	No

Toolkit Capabilities – Access Db2

	Node.js	PHP	Python
Retrieve properties for a DB2 Server	No	Yes	No
Retrieve index and statistics for a table	No	Yes	No
Retrieve tables and associated metadata in a database	No	Yes	No
Retrieve tables and associated privileges in a database	No	Yes	No

Toolkit Capabilities – Access ILE Applications and Artifacts

	Node.js	PHP	Python
Execute CL Commands		Yes	Yes
Call Programs *PGM	Yes	Yes	Yes
Call Service Programs *SRVPGM		Yes	Yes
Execute 'sh' commands (PASE utilities)	Yes	No	
Build/create a user space	Yes		
Retrieve information from a command definition object	Yes		
Retrieve information from a program object	Yes		
Retrieve Service Program Information	Yes		
Construct a new data queue object	Yes		
Send data to a data queue	Yes		
Retrieve information from a data queue	Yes		
Clear a data queue	Yes		

Toolkit Capabilities - Others

	Node.js	PHP	Python
Get PTF Information	Yes		
Retrieve TCP/IP attributes	Yes		
Retrieve Network interface information	Yes		
Retrieve user's authority to anobject	Yes		
Retrieve information about a user profile	Yes		
Retrieve information about users who are authorized to an object	Yes		

Unixcmd

Scott Klement – Unixcmd

- See https://community.common.org/webdev/blogs/temporary-admin1/2018/01/22/a-powerful-way-to-run-unix-and-open-source-tools-f

- Download at https://www.scottklement.com/unixcmd/

- Tool to run unix programs from RPG or CL in pase wie QP2SHELL API or Qshell via STRQSH or QSH commands
- Uses open access handler

RPG example

```
**FREE
dcl-f UNIX disk(1000) handler('UNIXCMDOA': cmd) usropn;
dcl-f QSYSPRT printer(132) usage(*output);
dcl-s cmd char(5000);
dcl-ds record len(1000) end-ds;
dcl-ds outrec len(132) end-ds;

cmd = 'cd /QIBM; ls';
open UNIX;

read UNIX record;
dow not %eof(UNIX);
   outrec = record;
   write QSYSPRT outrec;
   read UNIX record;
enddo;

*inlr = *on;
```

CL example

```
PGM

   DCL VAR(&REC) TYPE(*CHAR) LEN(1000)
   DCL VAR(&EOF) TYPE(*LGL)

   OPNPIPE CMD('cd /QIBM; ls') TYPE(*QSHELL)

   RCVPIPE RCD(&REC) EOF(&EOF)
   DOWHILE (&EOF *EQ '0')

      /* USING SNDUSRMSG FOR TESTING... YOU'LL WANT TO +
         REPLACE THIS WITH YOUR OWN CODE THAT USES THE DATA */

      SNDUSRMSG  MSG(&REC)

      RCVPIPE RCD(&REC) EOF(&EOF)
   ENDDO

   CLOPIPE

ENDPGM
```

Calling RPG from PASE

Call QSH from RPG/CL and vice-versa

- There is a special environment variable QIBM_QSH_CMD_OUTPUT.

- The value STDOUT let's you use the stdio ifs api’s to read out the the stdout in RPG.

- But you can also do it directly to a physical file. This is a technique we use a lot.

- ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) VALUE('FILE=lsout.txt')
- ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) VALUE('FILEAPPEND=lsout.txt')
- (fileappend will not overwrite but just add.).

Example

```
PGM
 MONMSG CPF0000
 ADDENVVAR  ENVVAR(QIBM_QSH_CMD_OUTPUT) +
            VALUE('FILE=/qsys.lib/shell.lib/lsout.file/+
            lsout.mbr')
 CRTPF      FILE(SHELL/LSOUT) RCDLEN(128)
 STRQSH     CMD('ls')
ENDPGM
```

- Content of the ls command will be in pf lsout.

Prestart job technique 1

You can speed up processing by using a prestart job,
which is a job that begins running when a subsystem is started.

When Qshell starts a new process, it will use a prestart job
if one is available. This improves performance,
because the system does not have to start a new job.

Prestart job technique 2

For example, the following command adds a prestart job to the QINTER subsystem description:

```
ADDPJE SBSD(QSYS/QINTER) PGM(QSYS/QP0ZSPWP)
 INLJOBS(10) THRESHOLD(5) ADLJOBS(10)
 JOBD(QGPL/QDFTJOBD) MAXUSE(1) CLS(QGPL/QINTER)
```

Prestart job technique 3

To make Qshell use a prestart job, place a value of Y

in the environment variable QSH_USE_PRESTART_JOBS.

Use the export command so that child processes will also use prestart jobs:

export -s QSH_USE_PRESTART_JOBS=Y

Special scripts 1

Special Scripts

When you begin a Qshell session,

it automatically executes the following

three script files, if they exist:

1/ The global profile file, /etc/profile

The system administrator uses this file to set

system-wide options for all users.

Special scripts 2

2/ A file named .profile After running /etc/profile,
Qshell looks in the users home directory for this profile,
which is used for personal customization.
(Yes, it begins with a period, and is pronounced "dot profile.")
The .profile file is a good place to define environment variables ,
including the ENV environment variable. Use the DSPUSRPRF command
to display your HOME directory.

Special scripts 3

3/ The file named in the ENV environment variable Qshell looks to see if the ENV environment

variable has a value. If so, and if that value is the name of an existing file, Qshell

executes the file. One of the most common uses of ENV is to define aliases ,

which are short names for a command.

Qshell runs these script files in the order given here, and in the current process.

Example to call RPG from qshell with parameters

RPG and COBOL programs receive parameters in a parameter list. Each passed parameter value is a null- terminated string.

Unpassed parameters cannot be addressed. Additional parameters, beyond the number declared, are ignored.

See example RPG source printargs

Example to call RPG from qshell with parameters

To extract the value of a parameter, use the %STR built-in function.

The %STR function is used in the ExtractParm subprocedure.

ExtractParm accepts a pointer to a parameter and passes that pointer to the %STR built-in function to access a null-terminated parameter value.

If a parameter value begins with a hyphen, the remainder of the parameter is processed as a string of options.

A parameter value that does not begin with a hyphen is assumed to be the argument of the last option that was found.

Example to call RPG from qshell with parameters

Since the options may be passed into the program in any sequence,

all of the following commands are equivalent and will produce the same output:

- /qsys.lib/shell.lib/printargs.pgm -b bval -c -f fval -2
- /qsys.lib/shell.lib/printargs.pgm -2cb bval -f fval
- /qsys.lib/shell.lib/printargs.pgm -f fval -c2b bval

Example to call RPG from qshell with parameters

```
Option selection
b: 1 "bval                                                  "
c: 1
C: 0
f: 1 "fval                                                  "
2: 1
```

STDIO

I sometimes use the QIBM_USE_DESCRIPTOR_STDIO environment variable.
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('Y')

This allows you to map the stdin/stdout/stderr as you wish.
You do not have to use ovrdbf or ovrprtf and can communicate with qp2term (pase) or qp2shell (qsh) directly in both directions.

The stdin = ifs read/write to file descriptor 0
The stdout = ifs read/write to file descriptor 1
The stderr = ifs read/write to file descriptor 2

STDIO example – NBRLINES (demo)

start qsh or qp2term

ls * | /QSYS.LIB/SHELL.LIB/NBRLINES.PGM

this will list the directory content and send it to rpg program nbrlines who will read the stdin and just add line numbers and write it back to the unix shell.

Check that your session (config) and job CCSID match !

Remember unixcmd

Scott Klements UNIXCMD works basically the same he just uses a pipe/spawn. This means he starts a second thread in the job to the unix command and then connects to the stdin, stdout and stderr through the pipe created.

I prefer to use it directly.

I have some CGI programs that just write to stdout and read from stdin instead of using all the CGI-api's. It runs much faster.

QzshSystem() API

To use the QzshSystem() API, you need to first open three temporary files and verify that those files get descriptor numbers 0, 1, and 2 before calling the API.

You must close the files before the program ends.

The following is the source for RPG program OPENSTDIO to open the three standard I/O file descriptors stdin (0), stdout (1), and stderr (2).

You should call this program as the first step in your job. If any other activity in the job opens one of these file descriptors incorrectly, this program will fail and your QSHELL or Java call might not work correctly.

The main procedure of the program ends with some rudimentary code to report the error; it uses a DSPLY operation and then it calls the *PSSR subroutine with ENDSR *CANCL which causes an exception to be sent to the program's caller. You may want to replace this code with a better mechanism for reporting the error.

OPENSTDIO – Part 1

```
h thread(*serialize) bnddir('QC2LE') dftactgrp(*no)

D O_CREAT         C                   x'00000008'
D O_TRUNC         C                   x'00000040'
D O_RDONLY        C                   x'00000001'
D O_WRONLY        C                   x'00000002'
D O_RDWR          C                   x'00000004'

D O_ACCMODE       c                   %BITOR(O_RDONLY
D                                          : %BITOR(O_WRONLY
D                                                 : O_RDWR))

D S_IRUSR         C                   x'0100'
D S_IROTH         C                   x'0004'
D S_IWUSR         C                   x'0080'
D S_IWOTH         C                   x'0002'
```

OPENSTDIO – Part 2

```
D chk                pr                n
D   descriptor                        10i 0 value
D   mode                              10i 0 value
D   aut                               10i 0 value
D   other_valid_mode...
D                                     10i 0 value

D ok                 s                 n
```

OPENSTDIO – Part 3

```
/free
    // Validate or open descriptors 0, 1 and 2
    ok = chk (0
          : 0 + O_CREAT + O_TRUNC + O_RDWR
          : 0 + S_IRUSR + S_IROTH
          : 0 + O_RDONLY)
    and  chk (1
          : 0 + O_CREAT + O_TRUNC + O_WRONLY
          : 0 + S_IWUSR + S_IWOTH
          : 0 + O_RDWR)
    and  chk (2
          : 0 + O_CREAT + O_TRUNC + O_WRONLY
          : 0 + S_IWUSR + S_IWOTH
          : 0 + O_RDWR);
```

OPENSTDIO – Part 4

```
// If the descriptors were not all correct,
    // signal an exception to our caller
    if not ok;
        dsply ('Descriptors 0, 1 and 2 not opened successfully.');
        exsr *pssr;
    endif;
    *inlr = '1';

    begsr *pssr;
    endsr '*CANCL';
   /end-free
```

OPENSTDIO – Part 5

```
P chk             b
D chk             pi              n
D   descriptor                   10i 0 value
D   mode                         10i 0 value
D   aut                          10i 0 value
D   other_valid_mode...
D                                10i 0 value

D open            pr             10i 0 extproc('open')
D   filename                       *   value options(*string)
D   mode                         10i 0 value
D   aut                          10i 0 value
D   unused                       10i 0 value options(*nopass)
```

OPENSTDIO – Part 6

```
D closeFile       pr            10i 0 extproc('close')
D   handle                      10i 0 value

D fcntl           pr            10I 0 extproc('fcntl')
D   descriptor                  10I 0 value
D   action                      10I 0 value
D   arg                         10I 0 value options(*nopass)

D F_GETFL         c                   x'06'

D flags           s             10i 0
D new_desc        s             10i 0

D actual_acc      s             10i 0
D required_acc    s             10i 0
D allowed_acc     s             10i 0
```

OPENSTDIO – Part 7

```
/free
    flags = fcntl (descriptor : F_GETFL);
    if flags < 0;
      // no flags returned, attempt to open this descriptor
      new_desc = open ('/dev/null' : mode : aut);
      if new_desc <> descriptor;
        // we didn't get the right descriptor number, so
        // close the one we got and return '0'
        if new_desc >= 0;
          closeFile (new_desc);
        endif;
        return '0';
      endif;
```

OPENSTDIO – Part 8

```
else;
        // check if the file was opened with the correct
        // access mode
        actual_acc = %bitand (flags : O_ACCMODE);
        required_acc = %bitand (mode : O_ACCMODE);
        allowed_acc = %bitand (other_valid_mode : O_ACCMODE);
        if  actual_acc <> required_acc
        and actual_acc <> allowed_acc;
          return '0';
        endif;
      endif;

      return '1';
   /end-free
  P chk           e
```

QzshSystem - example

- To compile CRTPGM PGM(libraryname/TEST) MODULE(libraryname/TEST) BNDSRVPGM((QSHELL/QZSHAPI))

```
H DEBUG(*YES)
DQzshSystem       PR            10I 0 ExtProc('QzshSystem')
D                                 *   value Options(*String)
DCommand          s             44A
D rc_qzsh         s             10I 0
 /free
   Command = 'pr -t ' + '/home/test.txt' + ' | ' +
   'Rfile -wQ qprint';
   rc_qzsh = QzshSystem(Command);
 /end-free
C                   seton                                        lr
```

Error handling

Unix systems (which QShell attempts to emulate) do not do error handling the same way as IBM i. Unix programs work like this:

1) Every program has an "exit status", programmers can set this however they like, but by convention, an exit status of zero means "success", and anything else implies some sort of error.

2) Error messages are typically printed to the "stderr" (standard error) data stream that, by default, is printed on the screen.

3) There are a few exceptions that are detected by the OS rather than the program. These are uncommon, but are sent as "signals".

That is very different from the model used in native IBM i applications. In native programs, an exception is generated by sending a message, and that message gets logged to a program message queue. All of the program message queues in a job, put together, make up the "job log." Unix doesn't work that way.

Error handling

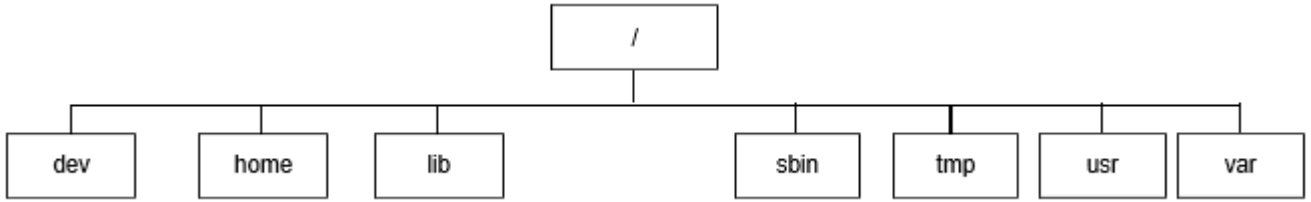
The MONITOR opcode will not catch an error in a Unix program, because MONITOR is looking for exception messages -- it's designed for IBM i. And errors won't be in the job log because that's not how Unix programs work, Unix systems don't even have a job log.

QIBM_QSH_CMD_ESCAPE_MSG envvar **tells QShell to send an escape message when the exit status is nonzero**. This allows you to use MONMSG or MONITOR to capture a failure !

Merci de votre attention

Setting up the OSS Ecosystem using ACS

The directory structure before install

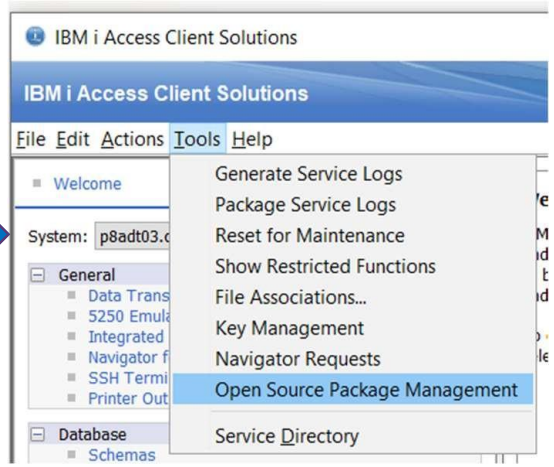


Directory	Description
bin	Commands
dev	Device files
etc	Configuration files
home	User home directories
lib	Libraries
pkgs	Package files / commands
sbin	Privileged commands
tmp	Temporary files
usr	Utilities & applications
var	Variable files

Bootstrapping OSS

- Bootstrapping is the process of installing utilities and repository definitions to enable the system with the necessary commands for managing open source packages

Version 1.1.8 or later of ACS

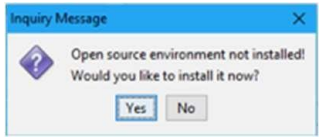


Connect to the system via SSH

Supports management of packages in

NOTE: An alternate installation method would be to download the SQL bootstrap script from f tp://public.dhe.ibm.com/software/ibmi/ products/pase/rpms/bootstrap.sql and run it against your IBM i

Bootstrap not currently installed



Repository definition

- The RPM packages reside in a repository that is publicly accessible
- The definition of the repository is located in the
- `/QOpenSys/etc/yum/repos.d` directory
  - The repository file for the IBM RPM pile is ibm.repo

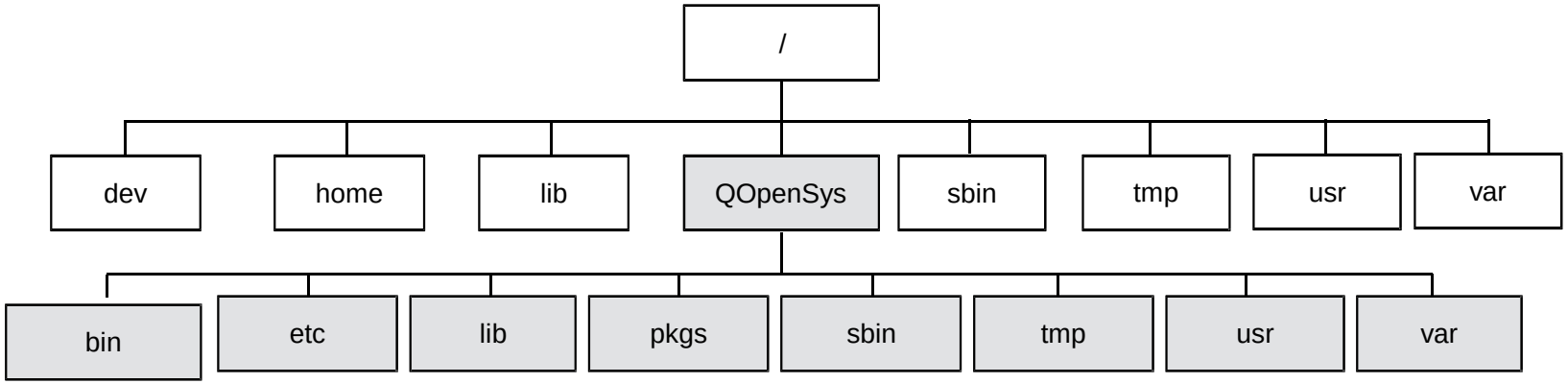
- `[ibm] name=ibm`
- `baseurl=http://public.dhe.ibm.com/software/ibmi/products/pase/rpms/repo enabled=1`
- `gpgcheck=0`

Note: it is possible to use a local repository by downloading the files from the indicated FTP site and then uploading them to a directory on the system. The `'baseurl'` would change to indicate `'file'` and the path to the directory of RPMs.

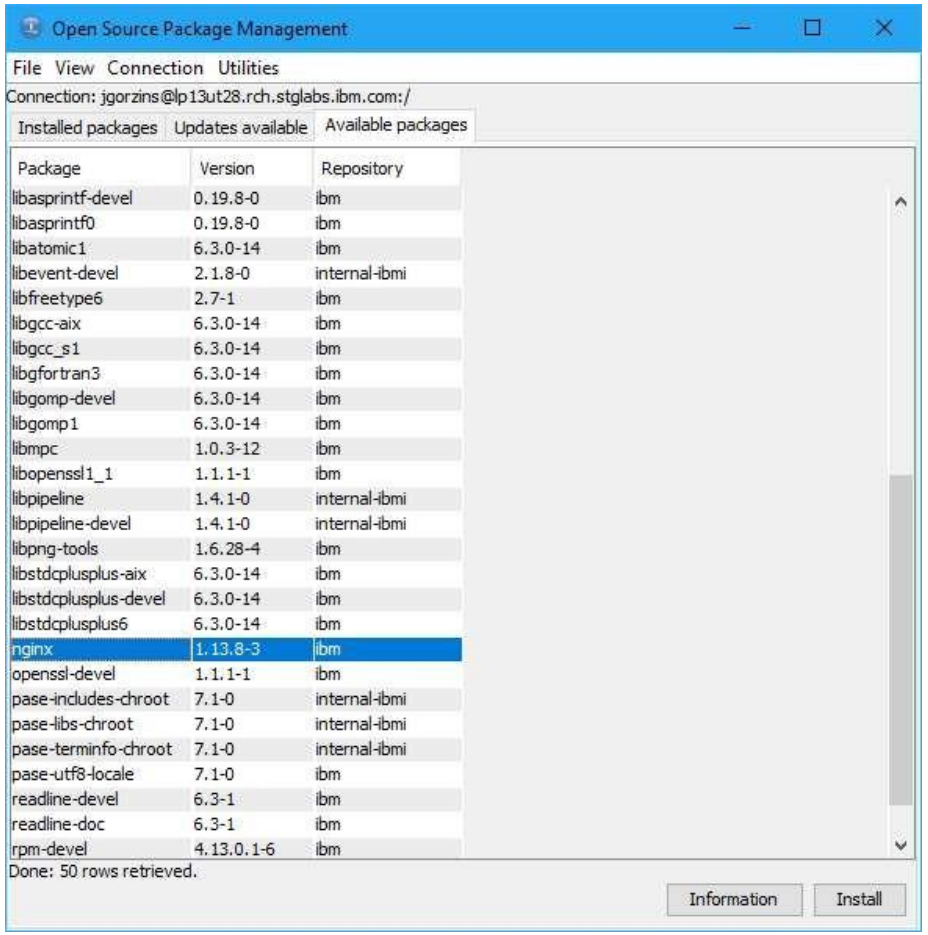
Additional note: ACS has support for cloning the repository to a local server

The directory structure

After installing the Open Source bootstrap

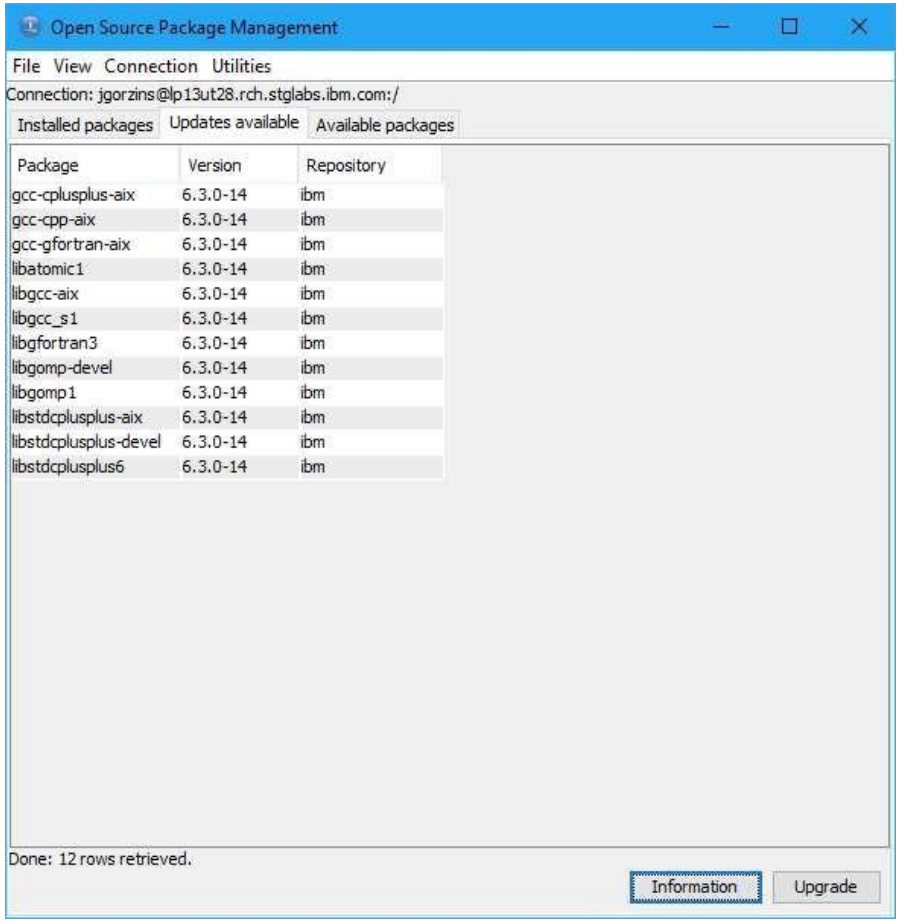


Install new software ACS




```
yum install <package>
```

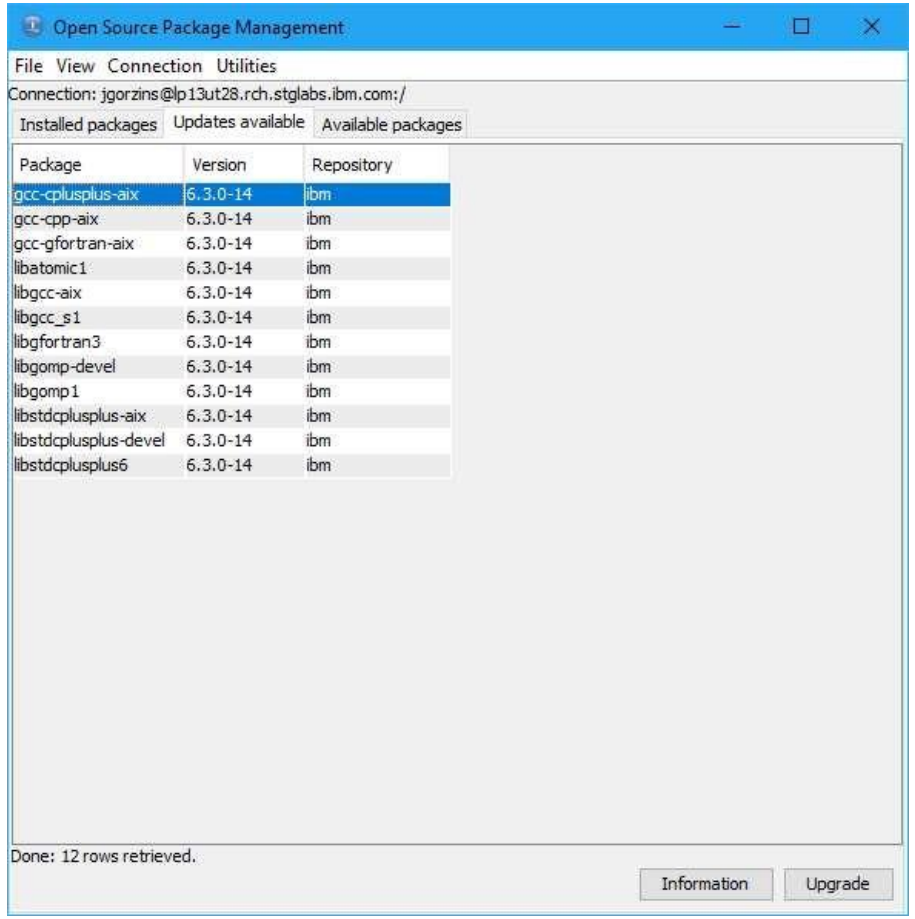
Check for upadtes ACS




```
yum list upgrades
```

zend.com

Perform an update ACS




```
yum upgrade <package>
```

```
yum upgrade
```

Security Vulnerability! Uh Oh!!!

- https://www.cvedetails.com/cve/CVE-2018-1000007/

Vulnerability Details : CVE-2018-1000007

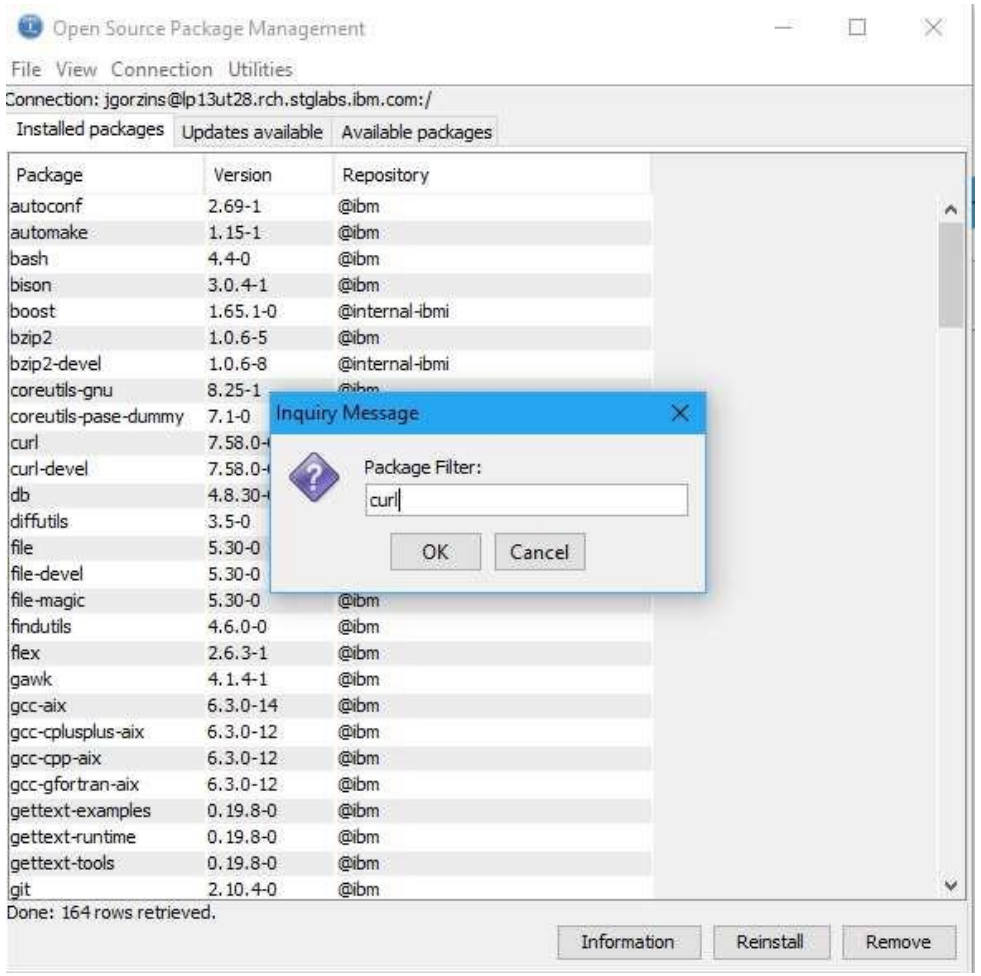
libcurl 7.1 through 7.57.0 might accidentally leak authentication data to third parties. When asked to send custom headers in its HTTP requests, libcurl will send that set of headers first to the host in the initial URL but also, if asked to follow redirects and a 30X HTTP response code is returned, to the host mentioned in URL in the `Location:` response header value. Sending the same set of headers to subsequest hosts is in particular a problem for applications that pass on custom `Authorization:` headers, as this header often contains privacy sensitive information or data that could allow others to impersonate the libcurl-using client's request.

Publish Date : 2018-01-24 Last Update Date : 2018-03-20

– Products Affected By CVE-2018-1000007

#	Product Type	Vendor	Product	Version	Update	Edition	Language	
1	OS	Debian	Debian Linux	7.0				Version Details Vulnerabilities
2	OS	Debian	Debian Linux	8.0				Version Details Vulnerabilities
3	OS	Debian	Debian Linux	9.0				Version Details Vulnerabilities
4	Application	Haxx	Curl	7.57.0				Version Details Vulnerabilities

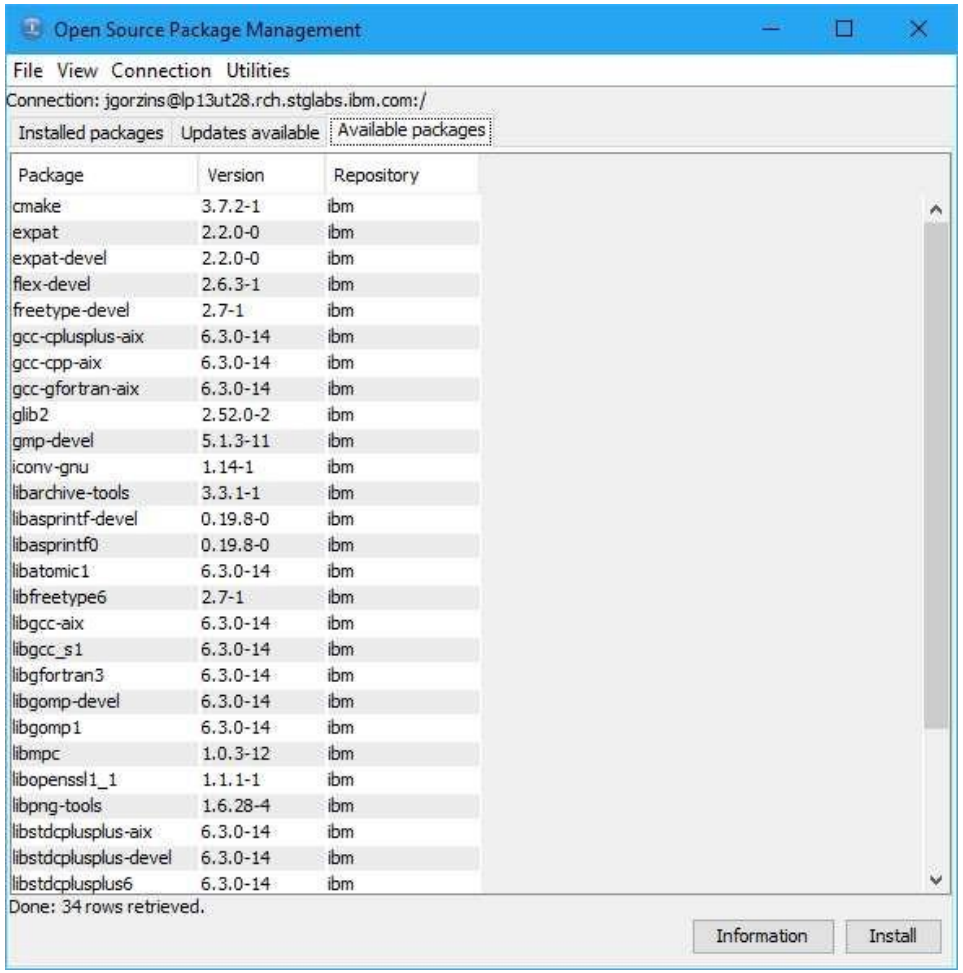
What's installed?: ACS




```
yum list <package>
```

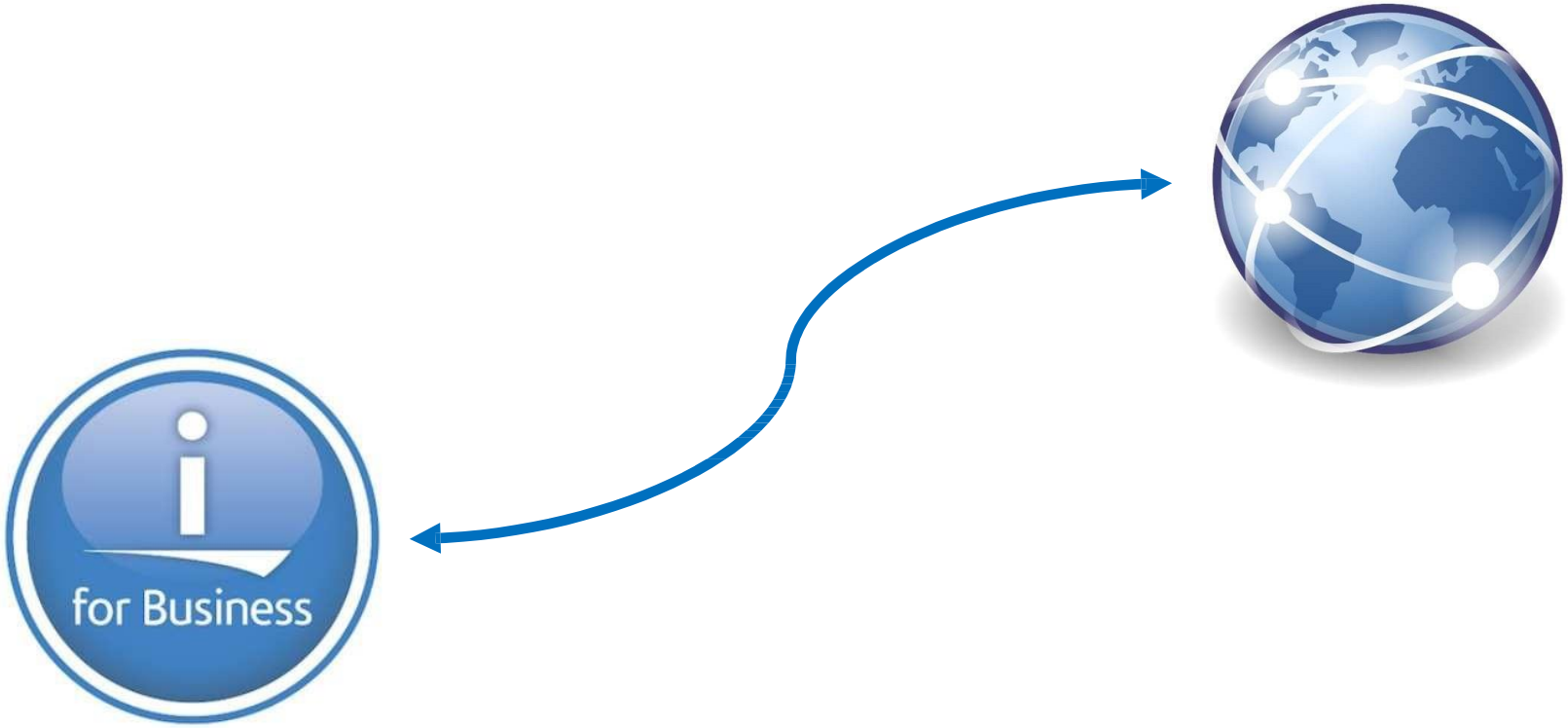
```
yum list '*searchword*'
```

What's available?: ACS

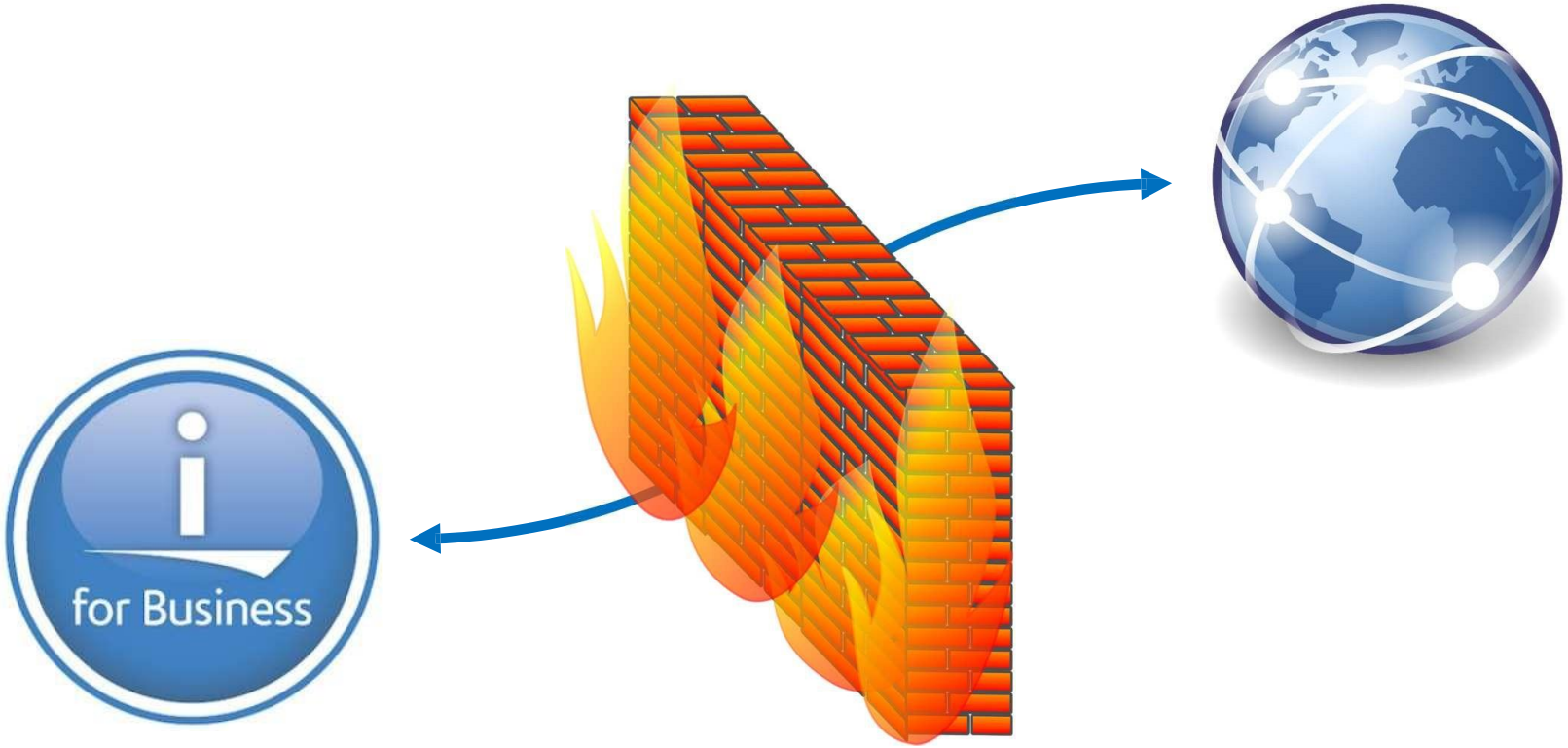



```
yum list available
```

Where does all this information come from?



Where does all this information come from?



Where are the RPMs (packages) hosted?

- Open-Source Software team builds .rpm files, then we put them out on the internet for you:

https://public.dhe.ibm.com/software/ibmi/products/pase/rpms/repo

Atlassian, Inc. (US) | https://bitbucket.org/ibmi/opensource/src/master/docs/yum/3RD_PARTY_REPOS.md 150% Search

opensource

- Source
- Commits
- Branches
- Pull requests
- Pipelines
- Deployments
- Issues
- Wiki
- Downloads
- Boards
- Settings

Third-party (non-IBM) repositories

The repositories listed on this page are not owned, managed, or supported by IBM. However, the repositories have been inspected and the software generally seems to be built with IBM-approved conventions for existing well in the IBM-delivered open source ecosystem.

Installation instructions

For each repository, this page lists a repo file and the contents for this repo file. In order to install this new repository, simply create the given repo file and populate it with the given contents using your favorite file editor.

Repository List

The i Doctor

Brought to you by: Jack Woehr

Software offered: lynx-dev (limited capabilities, for instance no https support). schily-tools (cdrecord, mkisofs, etc.)

repo file: `/QOpenSys/etc/yum/repos.d/the-i-doctor.repo`

repo file contents:

https://bitbucket.org/ibmi/opensource/src/master/docs/yum/3RD_PARTY_REPOS.md

Automating the Process

"How can I limit the RPMs available?"

- You need to host your own private RPM repository

"My systems can't access the internet!"

- You need to host your own private RPM repository **on your company's intranet**

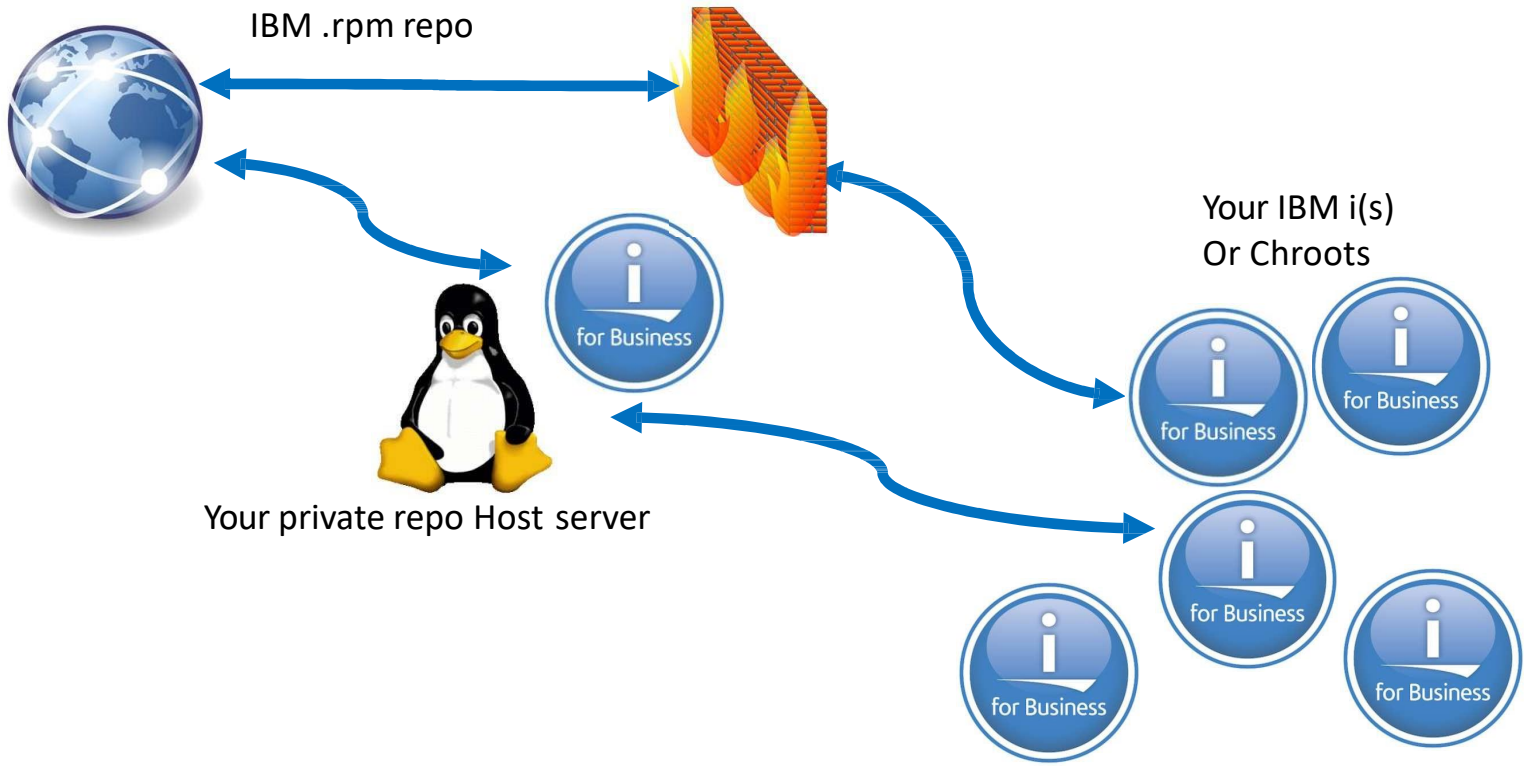
"I need to distribute to many systems"

- Easy to do with Yum – Script or Schedule a Job

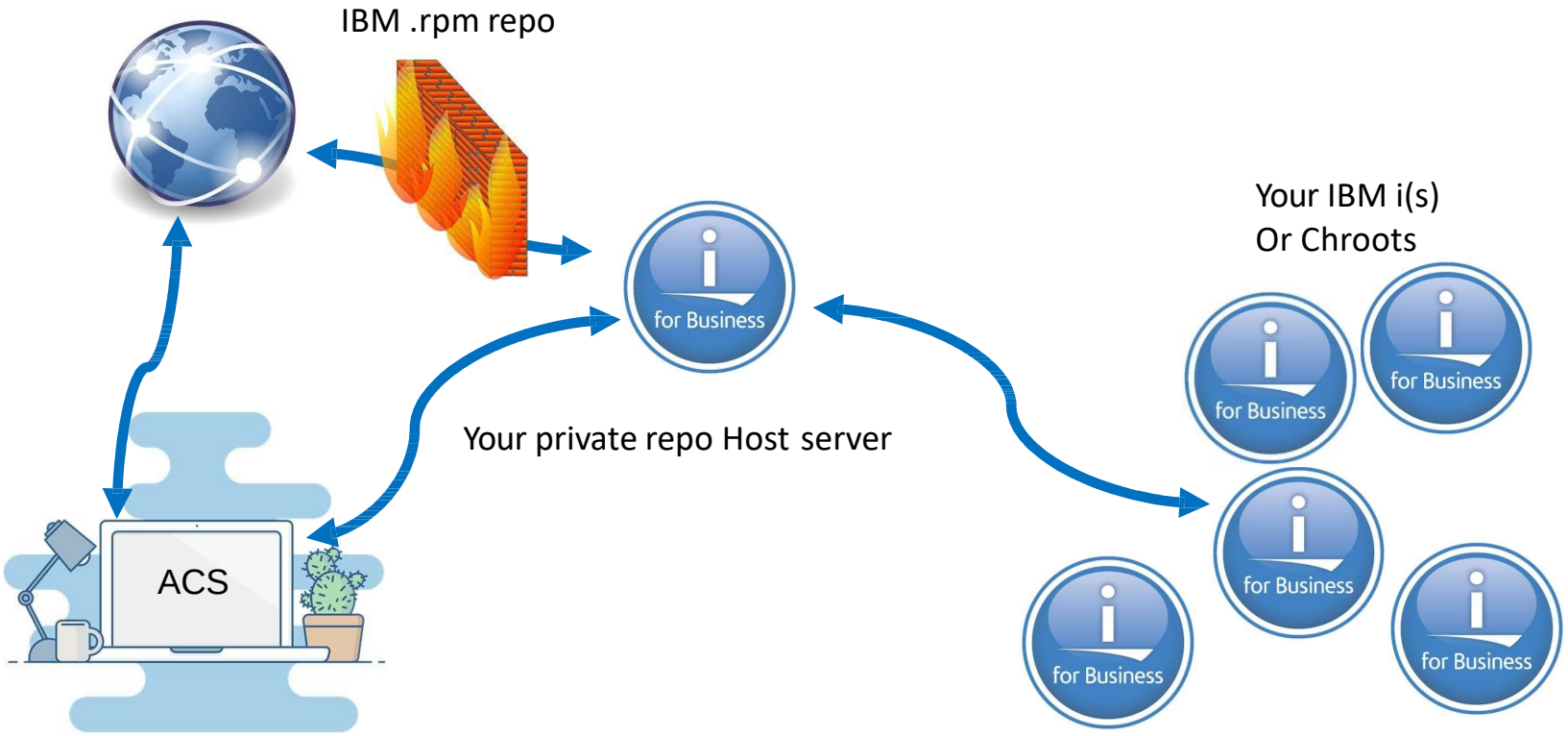
"That sounds really hard"

- Yes it does. But it is actually very easy. Let me show you...

Managing your RPMs company wide



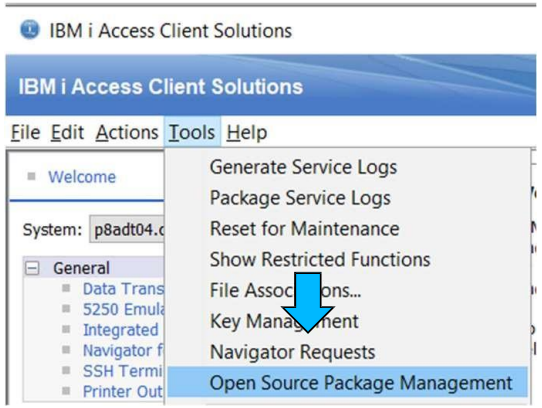
Managing your RPMs company wide

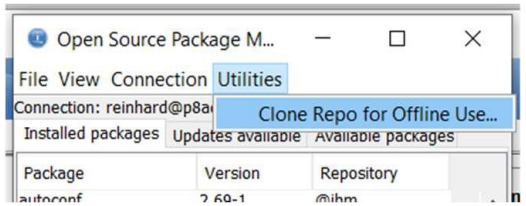


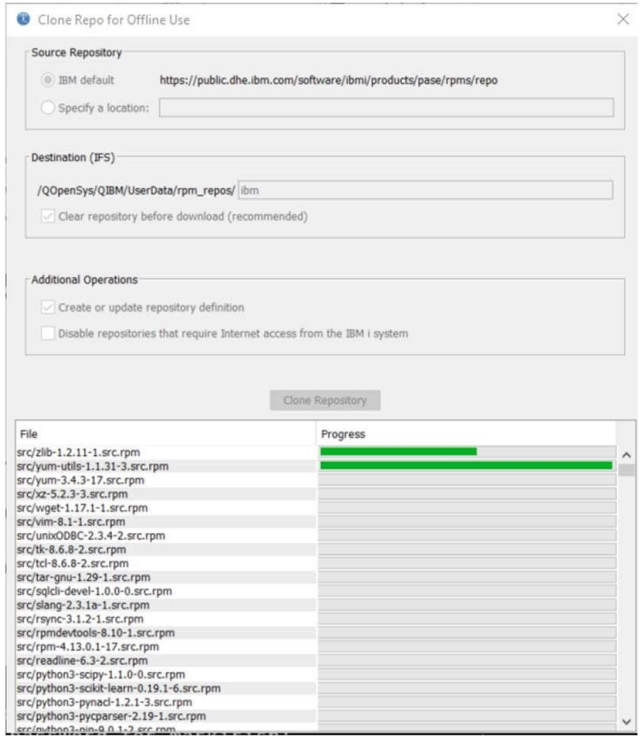
Distributing updates in 4 easy steps

1) Clone the IBM OSS repo to your Host server
2) Create your own repo
3) Point your IBM i systems to your repo
4) Automate…

Step 1: Clone IBM i OSS repo (using ACS)





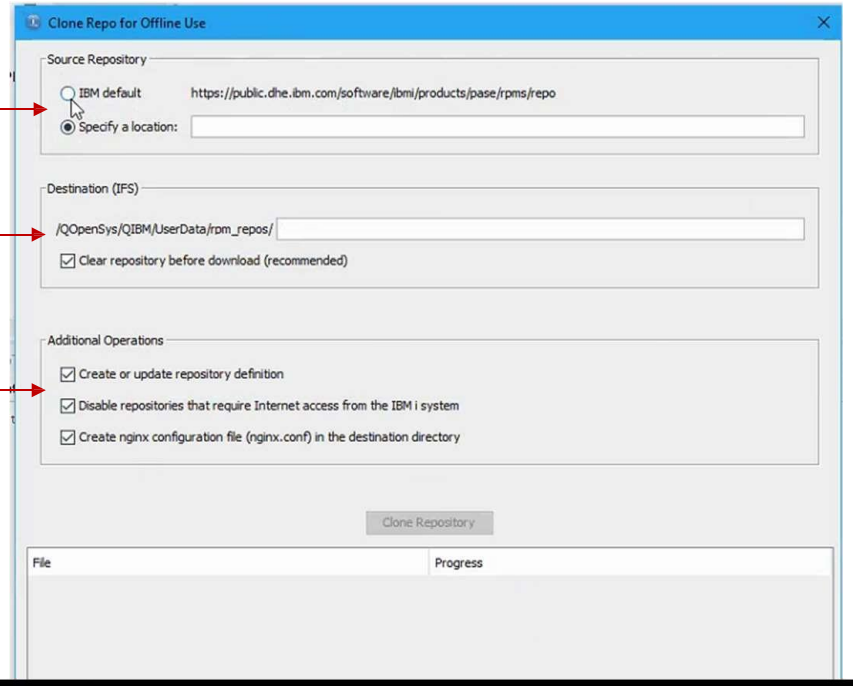


Step 2: Create your own repo (with ACS)

Copy from source repo

Create repo

Make repo accessible




```
-bash-4.4$ ls
logs/  nginx.conf  repodata/  startServer*  startServerBatch*  stopServer*
-bash-4.4$ ./startServer
```

Step 3: Point your IBM i systems to your repo

- Now, onto your IBM i systems…
- We have to set up our systems to point to our new repository

- **NOTE:** I use ssh to connect to my IBM i, and run bash script. **Some bash commands may be different than QP2TERM commands!**

Step 3: Point your IBM i systems to your repo

- This is where the ACS and non-ACS paths meet…
- On your IBM i, run the following command to point yum to the repo

```
yum-config-manager --add-repo <ip-address-where-hosted>/ibm
yum-config-manager --add-repo <ip-address-where-hosted>
```

Step 3: Point your IBM i systems to your repo

- When you run yum repolist, you should see your new, privately hosted repo!

192.168.2.40 - PuTTY

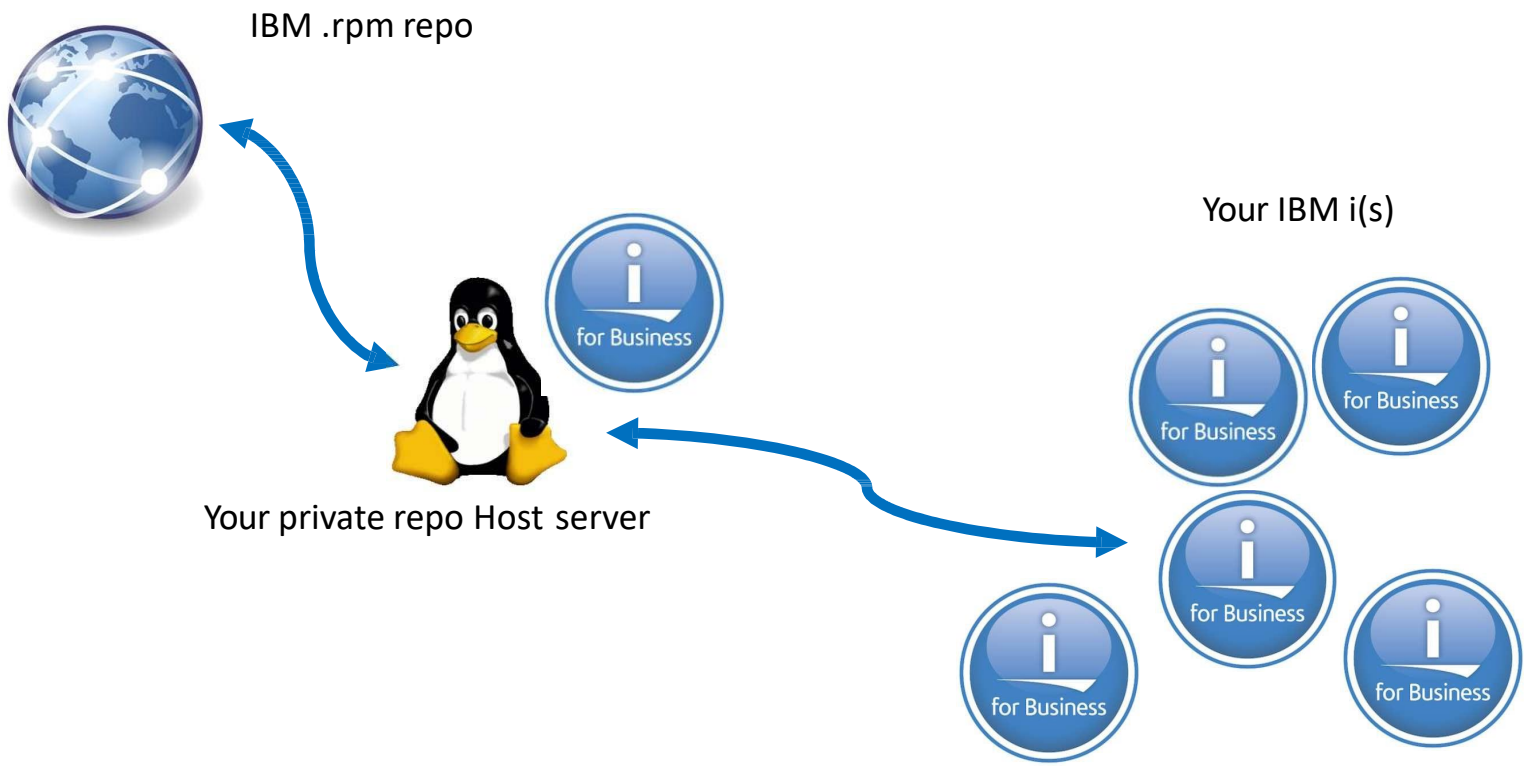
```
bash-4.4$ yum repolist
ibm
ibm/primary_db
repo id
ibm
repolist: 475
bash-4.4$
```

Step 3: Point your IBM i systems to your repo

Summary:

- We created a .repo file in /QOpenSys/etc/yum/repos.d
- We can now get our RPMs on our IBM i directly from our own repository

Automating updates



Utilities

- Various utilities are available for working in PASE including editors, package management tools, and source code control systems

- By installing the open source bootstrap (shown earlier) tools such as yum and rpm are available for manage other software packages

Function	yum command
Install a package	`yum install <package>`
Remove a package	`yum remove <package>`
Search for a package	`yum search <package>`
List installed packages	`yum list installed`
List available packages	`yum list available`
List all packages	`yum list all`

Utilities – Installing an Editor (example)

- Once the bootstrap has been installed, the `yum` command along with the repository definition are available to be used for installation of additional packages
- The '`yum repolist`' command can be used to validate the availability of the repository:

```
# yum repolist
repo id                                  repo name                          stat
                                                                            us
ibm                                      ibm                                231
repolist: 231
```

- A check can be made to see if a package with `nano` is available via the '`yum provides`' command:

```
# yum provides nano
nano-2.9.0-0.ppc64 : Small and friendly text editor
Repo          : ibm
```

Utilities – Installing an Editor (an example)

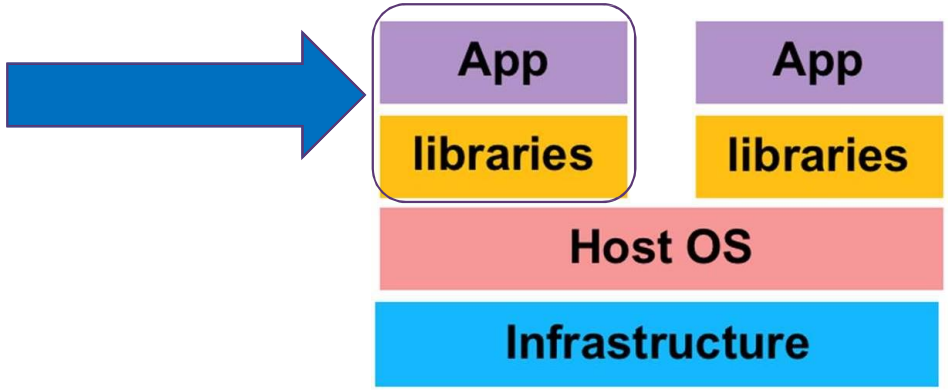
- The package can be installed via the '`yum install`' command:

```
# yum install nano
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package nano.ppc64 0:2.9.0-0 will be installed
--> Processing Dependency: lib:/Opens/pkgs/lib/libncurses.so.6(shr_64.o)(ppc64) for package: nano-2.9.0-0.ppc64
--> Running transaction check
---> Package libncurses6.ppc64 0:6.0-2 will be installed
--> Processing Dependency: ncurses-terminfo for package: libncurses6-6.0-2.ppc64
--> Running transaction check
---> Package ncurses-terminfo.ppc64 0:6.0-2 will be installed
→ Finished Dependency Resolution

Dependencies Resolved
================================================================================
 Package                    Arch           Version            Repository    Size
 ================================================================================
Installing:
 nano                       ppc64          2.9.0-0            ibm           598 k
Installing for dependencies:
 libncurses6                ppc64           6.0-2             ibm           318 k
 ncurses-terminfo           ppc64           6.0-2             ibm           582 k

Transaction Summary
================================================================================
Install        3 Packages
Total download size: 1.5 M
Installed size: 4.9 M
Is this ok [y/N]:
```

Containers on IBM i

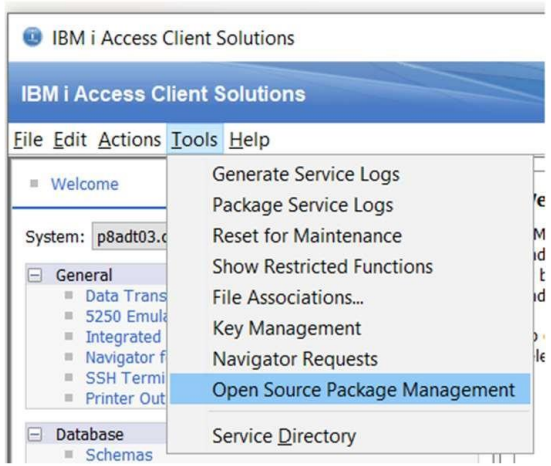
Overview

- Containers allow for the isolation of directories such that different users can have different environments that are dedicated to their usage.
- IBM i has the '`chroot`' capability available to enable establishing of what is often referred to as a 'chroot jail' – an isolation of directories and files.
- The following diagram provides a high-level view of chroot jail:



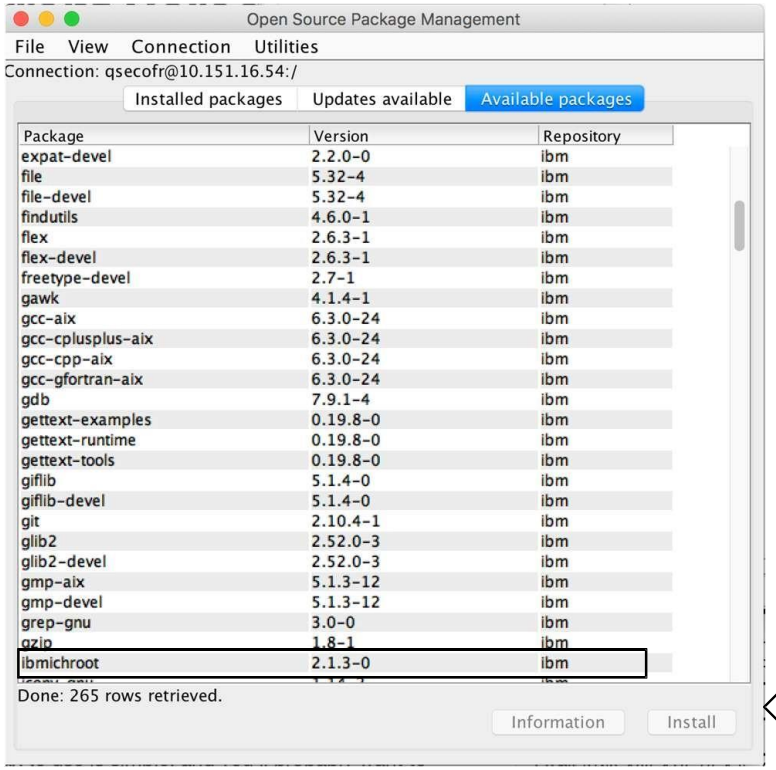
- In the above example there are two '`chroot`' jails that sit on top of the host operating system and the system infrastructure.
- Each "jail" can have their own software installed

Installing chroot support



Select package

CLI: yum install ibmichroot



Select install

Prompt with Actions to take

```
Setting up Install Processbin/yum  install 'ibmichroot'
-bash-4.4# clear;exec /QOpenSys/pkgs/bin/yum  install 'ibmichroot'

Resolving Dependencies
--> Running transaction check
---> Package ibmichroot.noarch 0:2.1.3-0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package             Arch            Version             Repository     Size
================================================================================
Installing:
 ibmichroot          noarch          2.1.3-0             ibm            20 k

Transaction Summary
================================================================================
Install       1 Package

Total size: 20 k
Installed size: 47 k
Is this ok [y/N]:
```

Diagnostic output

```
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : ibmichroot-2.1.3-0.noarch                                    1/1

Installed:
  ibmichroot.noarch 0:2.1.3-0

Complete!
```

Building the Container

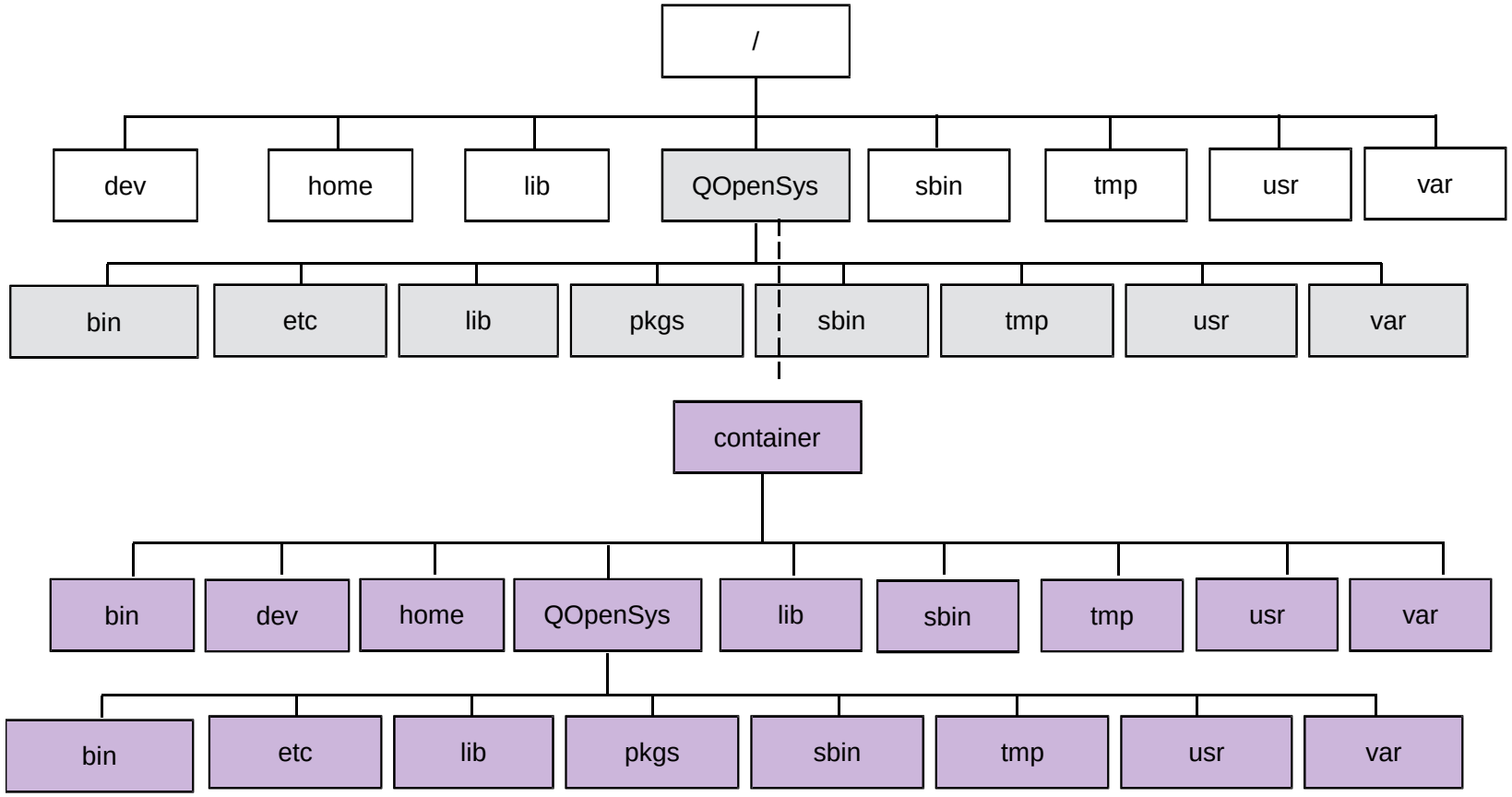
- Requirements:
  - Location of the chroot jail has to be under the /QOpenSys directory
  - The person running the 'chroot_setup' command needs to have *IOSYSCFG and *ALLOBJ

- Command to create the container:
  - chroot_setup /QOpenSys/<container name> minimal nls

```
 #####  #     # ######  ####### ####### #######
#     # #     # #     # #     # #     #    #
#       #     # #     # #     # #     #    #
#       ####### ######  #     # #     #    #
#       #     # #   #   #     # #     #    #
#     # #     # #    #  #     # #     #    #
 #####  #     # #     # ####### #######    #

 #####  ####### ####### #     # ######
#     # #           #    #     # #     #
#       #           #    #     # #     #
 #####  #####       #    #     # ######
      # #           #    #     # #
#     # #           #    #     # #
 #####  #######     #     #####  #

**********************
Live IBM i session (changes made).
**********************
/QOpenSys/testphp
/QOpenSys/testphp Does not Exist
```

The directory structure

After creating a container



Install Packages into Container (CLI)

Install a shell

```
The shell will be used for CLI
yum --installroot=/QOPenSys/<container> install bash
```

Install an editor

```
Determine package name:  yum provides nano
yum --installroot=/QOpenSys/<container> install nano
```

Install the open source language

```
yum --installroot=/QOpenSys/<container> install nodejs
```

Enter container

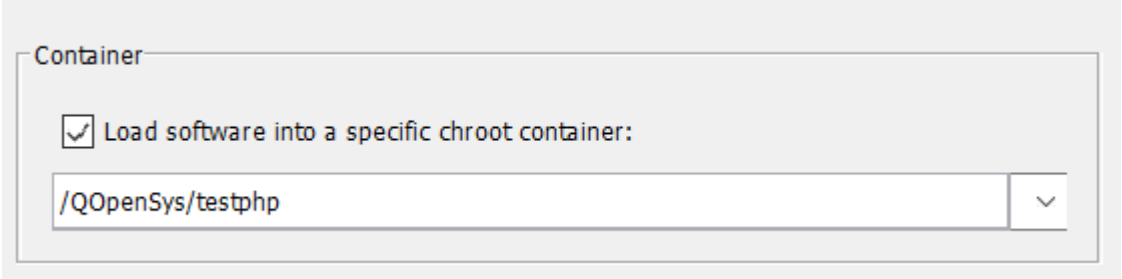
```
chroot /QOpenSys/<container> /QOpenSys/pkgs/bin/bash
```

Install db2 connector and toolkit (ILE)

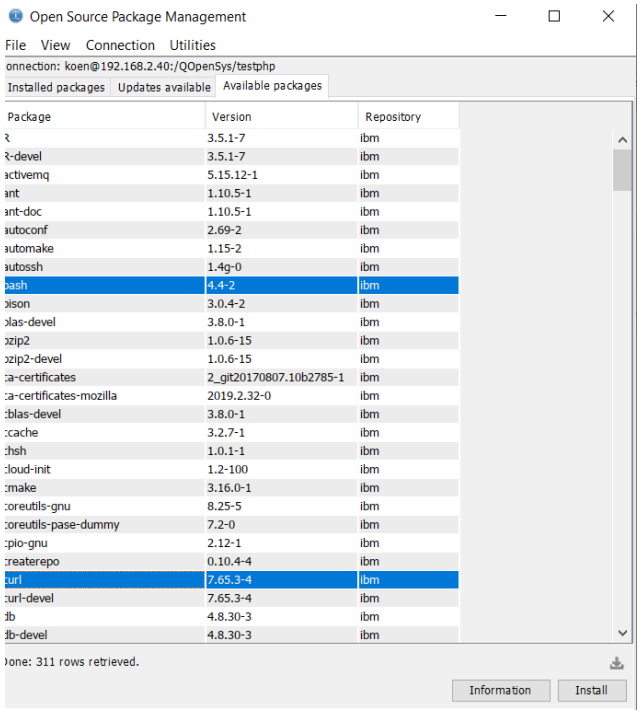
```
npm install idb-connector
npm install toolkit
```


Install Packages into Container (ACS)

Specify path for Container on connect



Select package from Available Packages tab



Select <Install>

Work in the Container

- The 'chroot' command can be used to enter the container:

```
chroot /QOpenSys/<container> /QOpenSys/usr/bin/bash
```

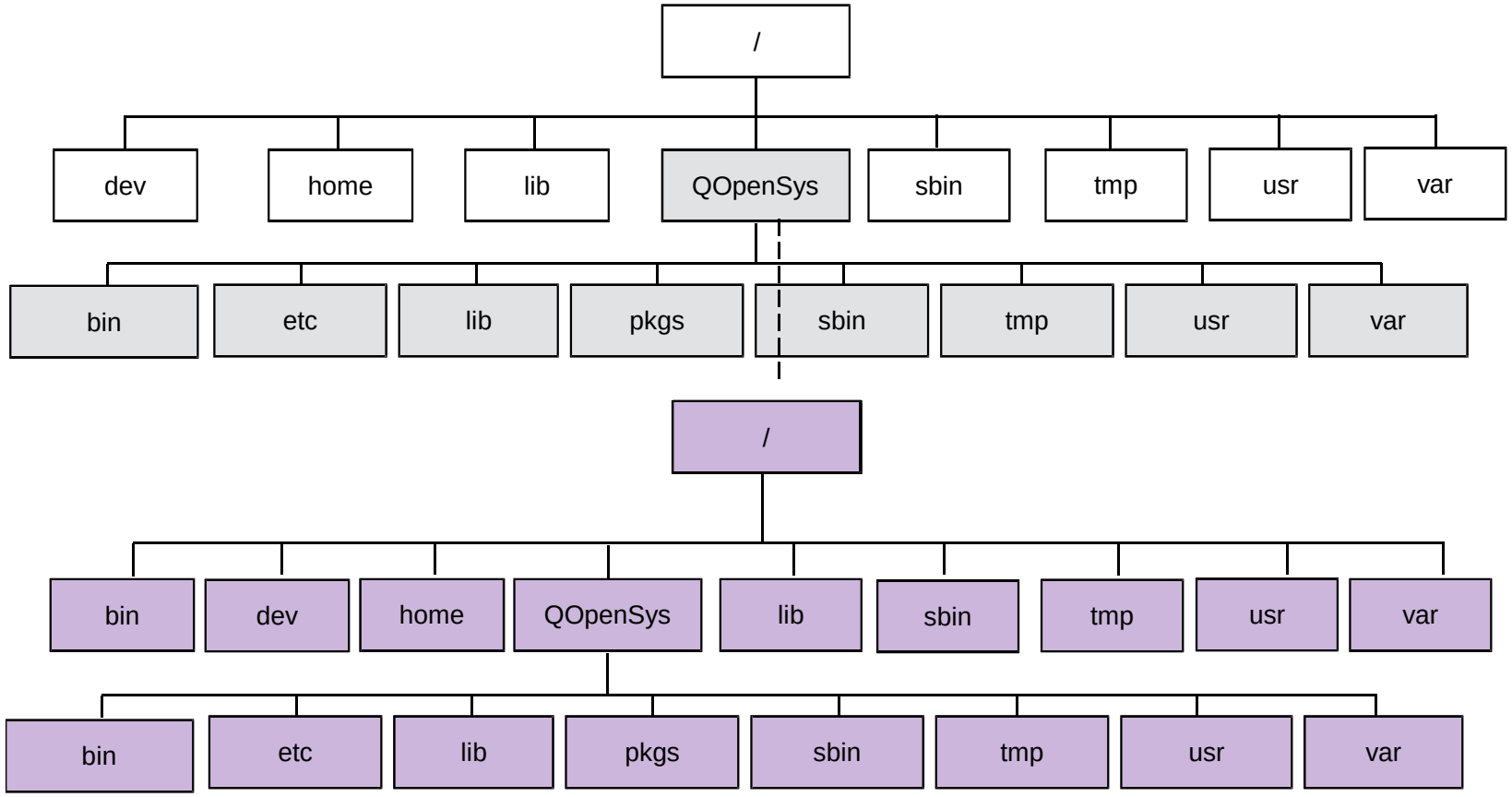
  - The first argument is the path to the container.
  - The second argument is the path (inside the container) of the first program to execute (typically a shell)

- Configure user profile to automatically enter the container:

```
CHGUSRPRF USRPRF(<USER>)
HOMEDIR('/QOpenSys/<container>/./home/<USER>')
```

The directory structure

Effect of entering the container



Managing Open Source on IBM i

Package Management Overview

- Common Tasks:
  - Install New Software
  - Check for Updates
  - Perform an Update
  - What's installed? Am I exposed to a CVE?
  - See What's Available

- Techniques
  - Access Client Solutions
  - CLI

rpm and yum Package Managers

- RPM: RPM Package Manager
  - Installs and manages individual packages
  - Works with .rpm files directly
  - Maintains the RPM database

- yum: Yellowdog Updater, Modified
  - Acts as a wrapper around RPM
  - Manages packages and dependencies automatically

RPM primary commands

Command	Usage
`rpm -i` or `rpm --install`	Install a package
`rpm -u` or `rpm --upgrade`	Upgrade a package
`rpm -e` or `rpm --erase`	Remove a package
`rpm -q`	Query the rpm database
`rpm -v`	Verify a package

YUM primary commands

Command	Usage
`yum install <package>`	Install a package
`yum update <package>`	Upgrade a package
`yum remove <package>`	Remove a package
`yum search <searchitem>`	Search a repository for a package
`yum list`	List packages
`yum info <package>`	Show details about a package
`yum provies <command>`	Find packages that delivers “command”

Managing Packages from inside a Container

- Install '`rpm`' and '`yum`' into the container:
  - `yum --installroot=/QOpenSys/<container> install rpm yum`
- Install '`yum-utils`' into the container:
  - `yum --installroot=/QOpenSys/<container> install yum-utils`

From inside the container

- Establish a repository definition:

  `yum-config-manager --add-repo`
  http://public.dhe.ibm.com/software/ibmi/products/pase/rpms/repo
- At this point you can use the typical package management commands (including ACS) from within the container

Building packages

Pre-Build Steps

- Install Open Source boot strap
- Install container support

```
yum install ibmichroot
```

- Establish container to perform the build in

```
chroot_setup /QOpenSys/phpbuild minimal nls
```

- Install headers into container

```
chroot_setup /QOpenSys/phpbuild chroot_includes
```

- Install bash into the container

```
yum install --installroot=/QOpenSys/phpbuild install bash
```

- Install yum and rpm into the container

```
yum install -installroot=/QOpenSys/phpbuild install yum
rpm
```

- Install developer tools group into the container

```
yum group -installroot=/QOpenSys/phpbuild groups install
"Developer Tools"
```

- Configure repository inside container

```
yum-config-manager --add-repo
http://public.dhe.ibm.com/software/ibmi/products/pase/rpm
s/repo
http://repos.zend.com/ibmiphp/ppc64/
```

Additional Installs

- bzip2-devel
- curl-devel
- freetype-devel
- libiconv-devel
- libintl-devel
- libjpeg-turbo-devel
- libpng-devel
- libsodium-devel
- libwebp-devel
- libxml2-devel
- libxslt-devel
- sqlite3-devel
- unixODBC-devel
- xz-devel
- gzip
- tar-gnu

Packages can be installed from inside or outside of the container

Performing the actual build

- Enter the container

```
chroot /QOpenSys/phpbuild /QOpenSys/pkgs/bin/bash
```

- Install the source

```
rpm -Uvh php-7.3.6-0.src.rpm
```

  - NOTE: The above command will create a SOURCES and SPECS directory in the user's home directory

- Perform the build
  - `cd SPECS`
  - `rpmbuild -ba php.spec`

- Copy the resulting RPMs to the repository

Open Source

What is open source software ?

“Software that gives users rights to run, copy, distribute, change and improve it as they see it, without them asking permission from or make payments to any external group or person”.

Mitre FOSS report 2002

Free as in freedom

- Freedom to study the code
- Freedom to improve the program
- Freedom to run the program anytime, for any purpose on any machine.
- Freedom to redistribute.

- Free Speech does not mean Free Beer !

Free OSS software

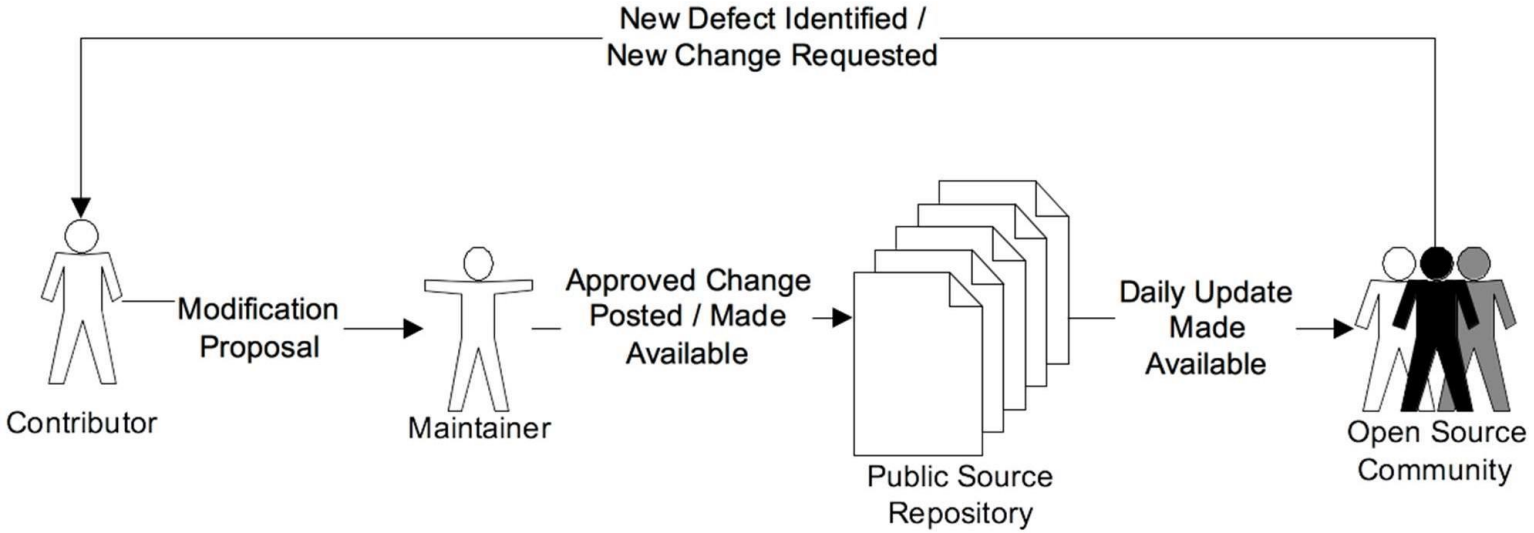
- Apache
- BIND
- Emacs
- FreeBSD
- Ghostscript
- Jakarta
- Jboss
- LaTex
- Linux
- MySQL
- Open Office
- Perl
- Samba
- Sendmail
- Snort
- Squid

Why OSS ?

- Customizable
- Improvable
- Redistributable
- Runs Everywhere, for everyone
- Transparency
- In many cases free of cost

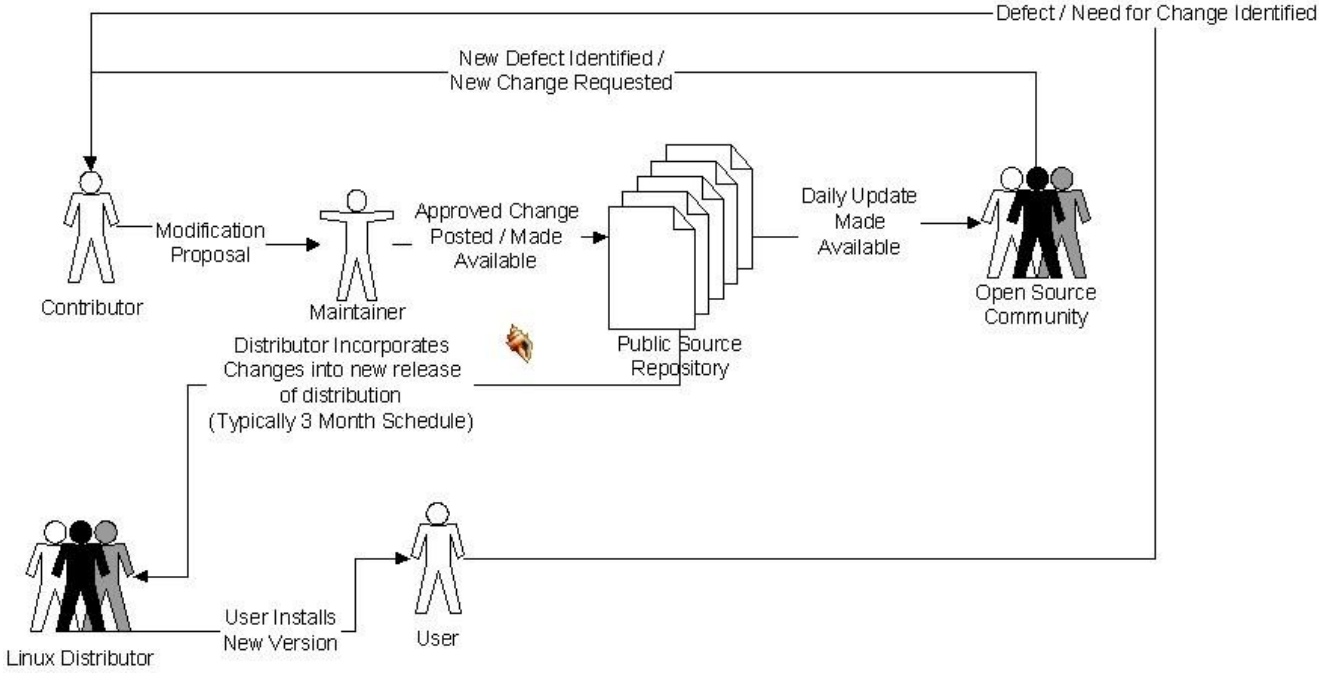
Understanding the open source process and distributions

- **Contributors**: Submit code changes to maintainers for consideration
- **Maintainers**: Ownership (from a maintenance viewpoint) of a particular component (application)
  - The maintainer is responsible for review of changes submitted by contributors, incorporation of accepted changes into the component, and releases of an updated component (application)



End-user influence on open source projects

- End-Users can report defects and request enhancements through the open source application's defect tracking system
  - Typically 'bugzilla'
- End-users have the "freedom" to review the open source, code their own changes, and submit them to the open source project maintainer.



Why open source development

- Collaboration is decentralized. Integration is controlled
- Too many resources to fail
- So many eyeballs looking at the code
- Self-motivated , self-assigned programmers.
- Large scale Peer Review
- User Driven requirements
- Developing in open community leads to innovation
- Develops open standards

Assessing open source solutions

- Evaluation/assessment of any solution is imperative to the successful implementation of a quality solution
- Steps in the evaluation process are really no different from those you would use with closed source (or commercial) solutions, just the time spent differs
  - **Step 1**: Identify candidate applications that may meet the business requirements
  - **Step 2**: Review existing reviews/evaluations of the application(s)
  - **Step 3**: Compare basic attributes of the application(s) with against specific requirements
  - **Step 4**: Perform an in depth analysis of the top candidate applications

- While the evaluation steps for proprietary and open source applications are the same the source of information for the identification and review steps are quite different
  - Proprietary solutions tend to be identified by the commercial vendor
  - Typically, the vendor provides documentation and literature on their product(s)
  - In the Open Source space, target applications are identified through a variety of mechanisms including search engines and well known open source repositories such as sourceforge.net and freshmeat.net

Open source evaluation

- Sources of information to assist with Identification and Review steps:
  - The Enterprise Open Source Directory (http://www.eosdirectory.com) provides an on- line catalog of open source projects
  - The opensourceCMS site (php.opensource.com) has a ratings page that provides customer ratings of popular CMS packages
  - Wikipedia has a list of free and open source applications by category (http://en.wikipedia.org/wiki/List_of_free_and_open_source_software_packages)
  - The Free Software Foundation maintains a Free Software Directory (http://directory.fsf.org/)
  - An Open Source Software Directory, compete with user ratings for the applications, is also available (http://www.opensourcesofsoftwaredirectory.com)
  - The InsideCRM web site (http://www.insidecrm.com) provides a list of the top 10 CRM packages. Rankings of CRM packages can also be found at the CRMSoftware360 website (http://www.crmsoftware360.com)
  - The wiki rankings available on the web are somewhat dated; however, you might want to take a look at the Top 5 open source Wiki engines article on the Tech Corner web site (http://www.benh.org/techblog/)
  - OSS Watch (http://www.oss.watch.ac.uk) can be a great resource for obtaining advice and guidance on the use, development and licensing of open source software applications

Open source evaluation

- **Factors to consider**

Reputation: What is the application's reputation for performance and reliability

- Some of the sites on the previous slide provide information on what others have experienced with the application
- Reputation of an application is often directly related to its popularity
  - While it's hard to gauge the popularity of a particular open source application there are some metrics that can be used to anecdotally determine the popularity such as download counters for the product as well as site counters for Internet based applications

Ongoing effort: Is there an ongoing effort to continue development of the application including both defect resolution as well as incorporating enhancements and new functionality?

- Places to look for this information include the application web site
- Another good place to look would be the sourceforge web site

Standards: Does the application adhere to a documented set of standards?

- Is there a requirement for the solution to be able to integrate with other applications?
  - What are the integration capabilities of the application under review?

Open source evaluation

- **Factors to consider**

Documentation: What level of documentation is available for the open source application?

- Popular open source applications like the SaMBa file server and Apache web server have a wealth of documentation available including both open source as well as commercial resources
- For less popular open source applications it is possible that documentation may lag release of product versions

Versioning: When was the last stable version of the application released?
Ensure that there is an ongoing development effort for the application

- Be aware that version numbers in open source don't necessarily follow the conventions of commercial applications
  - As an example, often times when evaluating commercial applications customers will tend to stay away from those applications whose version numbers end in a zero (0) as that represents the first release of a new version of the application
  - This may not be the case in open source – where 1.0 usually represents the first release of a commercial application, version 0.1 may represent the first version of an open source application
  - Keep in mind there is no “standard” for version numbers in the open source space.

Licensing: Review and understand the licensing aspects of the product and that it meets the requirements

- Not all open source licenses are created equal

Open source evaluation

- **F<u>actors to consider</u>**

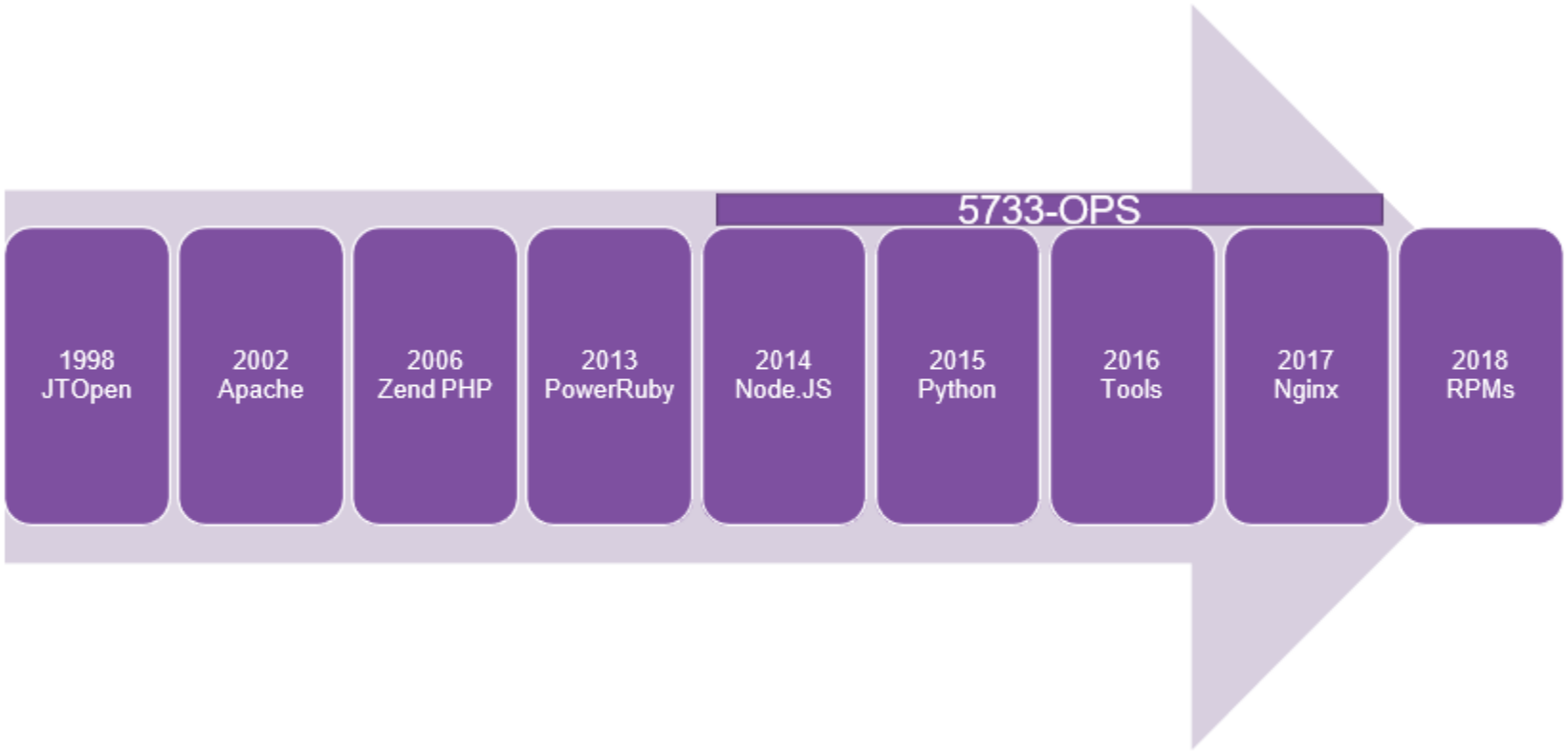
- Review of application literature should provide information necessary for the comparison of functions provided to those required
  - Keep in mind – this comparison will only be as good as the initial definition of the requirements to be satisfied

- Functional review should result in a subset (typically one or two) candidate applications migrating to the top of the "desired" list

- Here's the fun part!! Establish a test environment and kick the tires of those application(s) that you want to further review
  - Leverage the virtualization capabilities of the Power system to establish a Linux test LPAR (or KVM guest, or even a Docker container) for further evaluation of the application(s)

- Once the application face-off is complete, put the winning application into production.

Timeline open source on IBM i



IBM support

Linux
Subscription & Support

- Subscription & support for all major distributions of Linux including
- Linux system-level skills for multiple products
- Unmatched skills on IBM® System z®, IBM Power® and OEM Intel
- Focus on speed to resolution with direct access to IBM resources
- Basic, Enhanced & Premier support options available
- 99% TSS fix rate

Commercial OSS
Subscription & Support

- TSS can provide support solutions for the Red Hat & SUSE product portfolios
- Support for private cloud infrastructures running on multiple OpenStack distributions
- Software Defined Storage including Red Hat Ceph, Red Hat Gluster & SUSE Enterprise Storage
- Docker EE support available for IBM Power and System z platforms

Community OSS
Support

- Enterprise-class support for 100+ community versions of open source software
- IBM delivered L1/L2 support
- Available across x86, Power and System z
- Support includes diagnostics & virtually unlimited assistance with how-to, usage, configuration, installation, product compatibility and interoperability questions

Supported Packages include:

Apache HttpServer	*OpenJDK*
MariaDB	*Elasticsearch*
MongoDB	*Logstash*
MySQL	*Kibana*
PostgreSQL	*Cassandra*
ActiveMQ	*CouchDB*
Rabbit MQ	*Redis*
Tomcat	*Maven*
NGNIX	*Apigility*
WordPress	*GitLab*
SugarCRM	*CephFS*
Docker	*Kafka*
Kubernetes	*OpenLDAP*
Chef	*OpenSSL*
Puppet	*Zookeeper*
Spark	*Nagios*
Jenkins	*PHP*

ubuntu® redhat SUSE RogueWave SOFTWARE docker

More than just break/fix

80% of OSS support issues stem from either a lack of product knowledge, or something in the environment outside of the package [1]

80%

IBM Cloud Open Source Support includes diagnostics & virtually unlimited assistance with a wide variety of usage & how-to questions

TSS can be a resource for your development team….at any stage of the SDLC!

Interoperability Issues

- Product compatibility and interoperability questions
- Discuss interdependencies between OSS packages

Installation & Configuration

- Answer specific installation questions for documented functions
- Provide available configuration samples

Short Duration OSS Guidance

- We can provide advice on which OSS packages may be optimized or best suited for your solution

Community Engagement

- Rogue Wave & IBM participate in a wide variety of community projects and leverage as a resource

Our Solution Approach

- Our breadth of expertise allows us to take a holistic approach and provide support for the solution stack
- Review problems from a systems perspective

Additional Resources

- Our team can provide technical references to publications, such as redbooks or manuals and assist with interpretation of publications

[1] 2017 Open Source Support Report, Rogue Wave Software

The web is driven by open source

- Languages
  - PHP
  - Python
  - Ruby
  - Javascript
- Packages
  - JSON / XML
  - Swagger API framework
  - SOAP libraries
  - Web frameworks

- Application Framework/Servers
  - Apache Tomcat / TomEE
  - Jboss EAP
  - Greenfish
  - Rails
  - Epxress.js
  - Salis.js
  - Django
  - Bottle
  - Flask

- HTTP Servers
  - Apache HTTP Server
  - nginx
  - Eclipse Jetty

Open source skills are in high demand

- Open Source skills are the skills being sought after
  - Universities and trade schools offer wide-range of open-source related courses

https://www.tiobe.com/tiobe-index/

Mar 2022	Mar 2021	Change	Programming Language	Ratings	Change
1	3	^	Python	14.26%	+3.95%
2	1	v	C	13.06%	-2.27%
3	2	v	Java	11.19%	+0.74%
4	4		C++	8.66%	+2.14%
5	5		C#	5.92%	+0.95%
6	6		Visual Basic	5.77%	+0.91%
7	7		JavaScript	2.09%	-0.03%
8	8		PHP	1.92%	-0.15%
9	9		Assembly language	1.90%	-0.07%
10	10		SQL	1.85%	-0.02%
11	13	^	R	1.37%	+0.12%