Python Compatibility Dive: Don’t Let Strings Byte You in the Apps

Cory Burke
Principal Software Engineer

Samat Jain
Senior Software Engineer

splunk> .conf19

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or plans of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results may differ materially. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only, and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Agenda

1. Python Migration Overview
2. Specifying Python Runtime
3. Upgrading your app
   - Readiness Tools
   - Are You Affected?
   - Getting Started With Migration
   - Compatibility Tools
   - Nitty Gritty Pitfalls
   - Apps Built by Add-on Builder
4. Publishing
5. Resources
6. Q & A

Go find this deck and talk online!

FN1172: Python 2.7 End-of-Life: What it means for your deployment and apps

Python Compatibility For Admins

Learn more

Aditya Tammana
Product Manager | Splunk

Python Migration Overview

Admins & Developers

splunk> .conf19

Why Does This Matter?

1. Python 2.7 is End-of-Life on January 1, 2020.
2. As an **admin**, you will need to audit your environment for 8.0 incompatibilities – especially in apps.
3. As a **developer**, you will need to make your app compatible with Python 2.7 and 3.7.
4. Very soon, we will stop shipping Python 2.7.

What's Happening?

1. Enterprise 8.0 ships with Python 2.7 AND Python 3.7 runtimes
2. Splunk Web (the appserver) is Python 3.7 only. CherryPy 18.x is Python 3 ONLY.
3. Some features have been removed!

8.0 Prerequisites

Do this before upgrading!

Stop Using These

Advanced XML

- Removed in 8.0
- Deprecated 4 years ago

Splunk Web Legacy Mode

- `appServerPorts=0` in web.conf
- Deprecated 3 years ago

Update These to Support Python 3.7

Custom CherryPy Endpoints

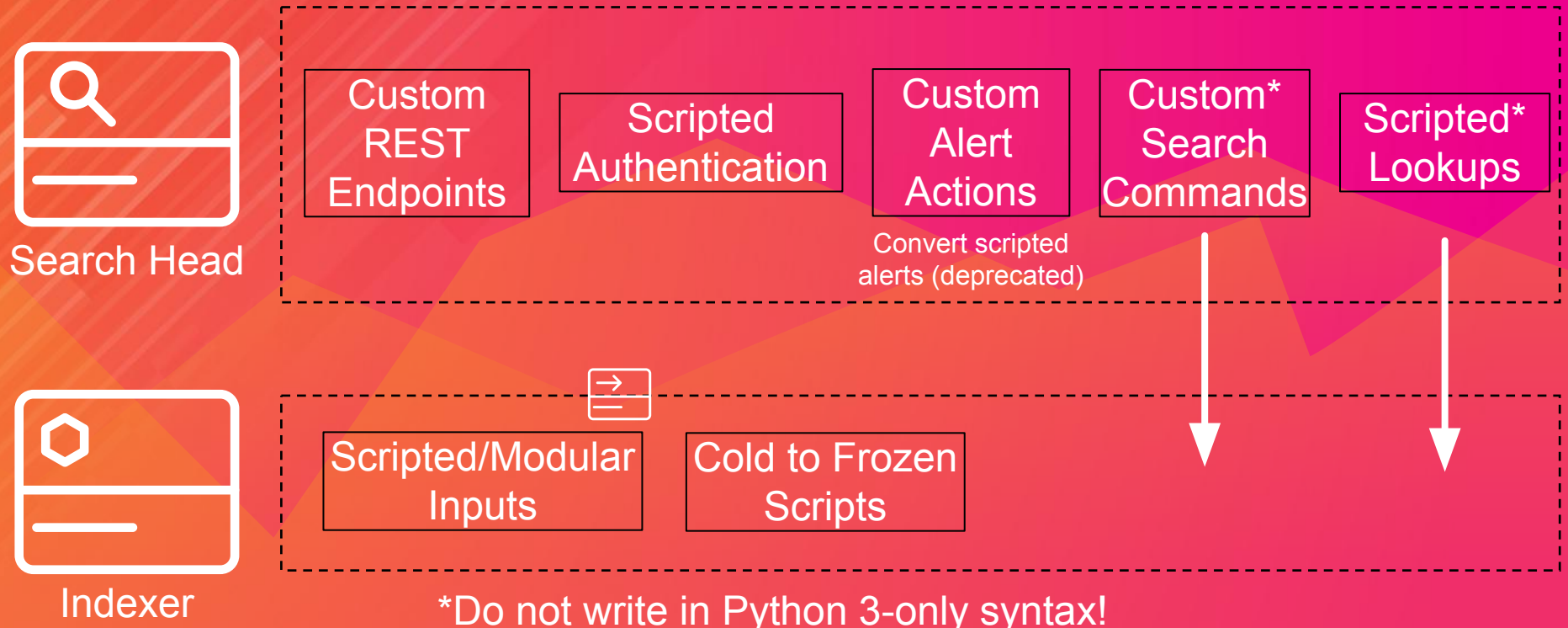
- AKA custom web controllers
- Make dual-compatible for easier upgrades

Custom Mako Templates

- Python can be wrapped in HTML using Mako
- Make dual-compatible for easier upgrades

Other Supported Python Scripts

Will work with Python 2.7 on Enterprise 8.0



*Do not write in Python 3-only syntax!

Will fail if indexer < 8.0

When should we start upgrading?

Start now!

There's treasure at the end of this, right?

2

Specifying Python Runtime

Admins & Developers

Why Does This Matter?

As a **developer,** you can specify which Python runtime to use on a per-script basis.

As an **admin**, you can specify which Python runtime to use across an entire instance (and for scripts in any apps on your instance).

What Should I Remember?

The **default** runtime in 8.0 is Python 2.7

*Except for Splunk Web (the appserver) and the CLI

Specify Python Runtime

Global Runtime Setting

Default Python 2

- Shipped OOB with 8.0
- All scripts without override will run Python 2

Default Python 3

- All scripts without override will run Python 3

Force Python 3

- All scripts will run Python 3, regardless of override
- Meant for those with strict support reqs

Script Runtime Setting

No specification

- Will run version specified globally

Python 2

- Will override any default global setting
- Cannot override default force setting

Python 3

- Will always run Python 3

3.1

Upgrading your app: Readiness Tools

Admins & Developers

splunk> .conf19

What's Happening?

Splunk will provide an **Upgrade Readiness App** to help prepare for the move to 8.0.

Why Does This Matter?

This app scans all other apps on an instance for **impacted components** due to the Python migration.

In large deployments with dozens of apps, this is a useful identification tool.

3.2

Upgrading your app: Are you affected and compatibility

Developers

splunk> .conf19

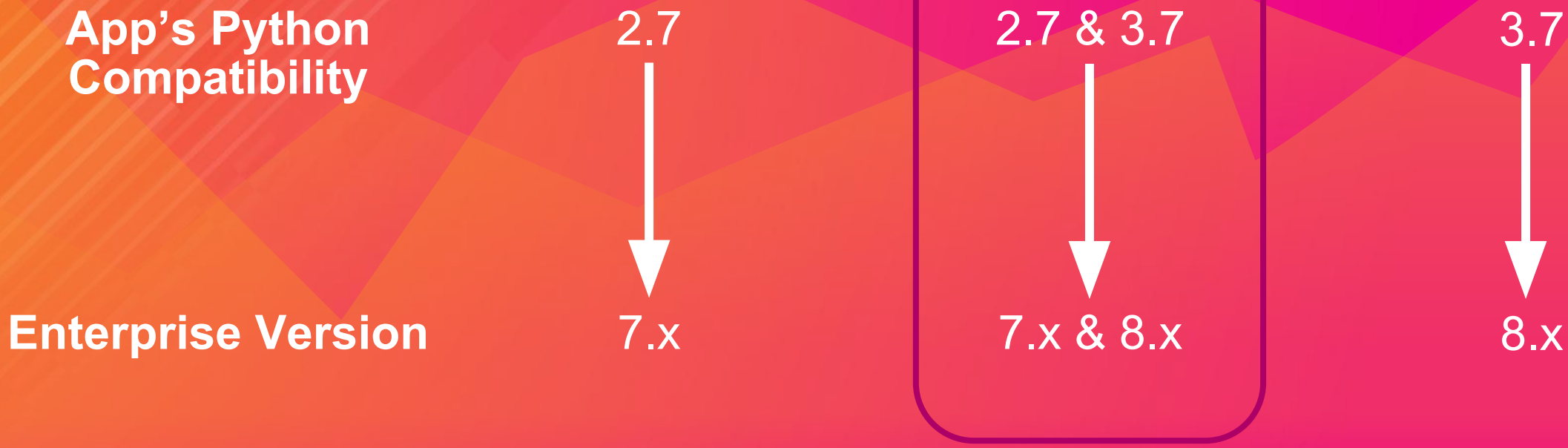
How Do I know if I'm Affected?

1. Do you have any of the following?
2. Mako templates
3. CherryPy Controllers/Custom REST Endpoints
4. Scripted:
   - Inputs
   - Modular Inputs
   - Custom Search Commands
   - Authentication
   - Lookups
   - Alert Action

What do we do now?

1. Do Nothing
   - Your app probably won't work with Splunk 8
2. Port for Python 3 syntax
   - Your app will not be backwards compatible
   - DO NOT DO THIS UNLESS YOU HAVE NO CHOICE
3. Port for Python 2 and 3 compatibility
   - Your app will continue to be backwards compatible
   - This is important for mixed version deployments
   - Customers can upgrade your app before migration

Enterprise vs. Python Compatibility



3.3

Upgrading your app: Getting started with migration

Developers

splunk> .conf19

How to get started upgrading your app

These apply once you have initialized a Splunk 8.0 sandbox

All at once

In server.conf, set:

```
python.version = force_python3
```

Restart Splunk.

Test and fix all the problems.

One component at a time

For each component (e.g. modular input), set:

```
python.version = python3
```

in the stanza for that component.

Restart Splunk.

Text and fix all the problems.

How do I get debug logs?

Use an $SPLUNK_HOME/etc/log-local.cfg file:

```
[splunkd]
category.ModularInputs=DEBUG

[python]
splunk = DEBUG
splunk.appserver = DEBUG
splunk.appserver.controllers = DEBUG
splunk.appserver.controllers.proxy = DEBUG
splunk.appserver.lib = DEBUG
splunk.pdfgen = INFO
splunk.archiver_restoration = INFO
```

Enable SplunkWeb startup logging (to splunkd.log) in web.conf. New for 8.0, don't use in production!

```
appServerProcessLogStderr = true
```

Logs go into $SPLUNK_HOME/var/log/web_services.log, web_access.log, python.log, and splunkd.log

3.4

Upgrading your app: Compatibility tools

Developers

splunk> .conf19

How do I make code Python 2 and 3 compatible?

What it looks like under each Python version

Python 2-only code:

```
from urllib import unquote
```

Python 3-only code (maybe you ran 2to3):

```
from urllib.parse import unquote
```

How do I make code Python 2 and 3 compatible?

"Defensively" code

```
try:   # Python 3
    from urllib.parse import unquote
except ImportError:  # Python 2
    from urllib import unquote
```

How do I make code Python 2 and 3 compatible?

Use 2-to-3 helper libraries, e.g. moved library import helper (recommended)

Using the six library:

```
from six.moves.urllib.parse import unquote
```

Using the future library:

```
from future.moves.urllib.parse import unquote
```

How do I make code Python 2 and 3 compatible?

Write “idiomatic” Python 3 with future (Python experts only!)

```
# TODO: Delete this when you no longer care about Python 2

from future.standard_library import install_aliases

install_aliases()


from urllib.parse import unquote
```

Automatically fixing code: libraries

six [recommended]

- One file library, easy to include in your app!
- Heavily used throughout Python community
- Not “idiomatic” Python 3 code, creates hard requirement on six

future

- Mostly “idiomatic” Python 3 code, does not create hard dependency on future
- Not to be confused with “__future__” built-into Python
- Complex directory structure, can be difficult to include in your app
- Recommended by Python.org
- Used internally in Splunk 8.0

Automatically fixing code: tools

2to3

- From Python upstream
- Idea: automatically convert Python 2 to Python 3 code
- Doesn’t really work, but introduces concept of “fixers”

modernize (uses six)

- Builds on 2to3, has fixers has will use the six library

futurize

- Builds on 2to3, has fixers that use the future library
- Divides fixers into “stage1” and “stage2”
- Used when porting Python 2 to 3 for Splunk 8.0 (to identify problem spots)

Automatically fixing code: fixers

Use fixers to find fix code automatically or find problem spots

E.g. Turn all print statements into print functions

```
$ futurize -f libfuturize.fixes.fix_print_with_import  py2.py
RefactoringTool: Refactored py2.py
--- py2.py      (original)
+++ py2.py      (refactored)
@@ -1 +1,2 @@
-print 'hello world'
+from __future__ import print_function
+print('hello world')
RefactoringTool: Files that need to be modified:
RefactoringTool: py2.py
```

Automatically fixing code: tips for using fixers

Strategy

- Run one fixer at a time
- Test, fix problems
- Commit changes (avoid batching code changes for different fixers together)
- Repeat

futurize

- “stage1” fixers are likely safe for you to use and let make automated changes, EXCEPT for absolute import fixer

Don’t rely on fixers to automatically fix for you. Use to identify problem spots.

- futurize’s “stage2” fixers caused more problems than worth it, took this approach for Splunk 8.0

Supporting Python scripts on Splunk versions

Latest Splunk 8.0 and latest supported maintenance releases

Included libraries:

- future, with working past/lib2to3
- six

For Python 2.7 in Splunk 8.0, future and six won't be upgraded. Locked at future 0.17.1 and six 1.12.0.

If you *MUST* support older Splunk

- Option 1: Include and use six in your app [recommended]
- Option 2: Include and use future in your app. Do not use "past" or "lib2to3".
- Option 3: Use no helper library, defensively code everything

Include your own copy--do not rely on Splunk Enterprise's copy!

Review the Splunk support policy to see which release versions are supported.

3.5

Upgrading your app: Nitty gritty pitfalls

Developers

splunk> .conf19

“String” world vs “bytes” world

Text vs binary data

Text aka “string” world

- Composed of characters, in the linguistic sense. Like “a” or “喂”
- Python 3, use str()
- Python 2, use unicode(), or str() if only concerned about ASCII/ANSI
- Use .encode() to convert to bytes world

Binary data aka “bytes” world

- An “encoding” defines what series of 0s and 1s (typically grouped into bytes, aka C chars) represents a glyph. A glyph may take more multiple bytes to represent.
- In ASCII, ‘a’ is 97, 0x61, or 01100001
- Python 3, use bytes(), bytearray()
- Python 2, can use bytes(), bytearray(), or str()
- Use .decode() to convert to string world

Bytes biting you in your Apps

Python 2, string world and bytes world can be the same if you're using ASCII or ANSI. If you weren't keeping track of it, separating the two, in practice, gets difficult.

Python 3, you must keep track of whether you're in string world or bytes world. They're incompatible. In practice, not hard.

Storing binary data as text, or text as binary data, causes exceptions.

For app compatibility with Splunk 8.0 and all earlier supported versions, you *must* code defensively when dealing with strings.

Getting text back from APIs, json.loads()

json.loads() can take bytes or strings. json.loads() assumes an encoding (UTF-8 for Splunk's Python)

```
>>> json.loads(b'{"hello": "world"}')
{'hello': 'world'}

>>> json.loads('{"hello": "world"}')
{'hello': 'world'}
```

Getting text back from APIs, json.dumps()

json.dumps() ONLY takes strings, no bytes!

```
>>> json.dumps({"hello": b"world"})
Traceback (most recent call last):
 File "<input>", line 1, in <module>
   json.dumps({"hello": b"world"})
 File "/usr/lib/python3.7/json/__init__.py", line 231, in dumps
   return _default_encoder.encode(obj)
 File "/usr/lib/python3.7/json/encoder.py", line 199, in encode
   chunks = self.iterencode(o, _one_shot=True)
 File "/usr/lib/python3.7/json/encoder.py", line 257, in iterencode
   return _iterencode(o, 0)
 File "/usr/lib/python3.7/json/encoder.py", line 179, in default
   raise TypeError(f'Object of type {o.__class__.__name__} '
TypeError: Object of type bytes is not JSON serializable
```

Getting text back from APIs, simpleRequest

Any app that uses simpleRequest will be bitten by bytes this way

- splunk.rest.simpleRequest makes a REST HTTP request — low level API for any request
- Always returns bytes - low-level API does not know what response will be
- Caller must convert binary data to text with decode():

```
# Works under Python 2 and 3
rest_response_content_raw = splunk.rest.simpleRequest(...)
if sys.version >= (3, 0):
    rest_response_content rest_response_content_raw.decode()
```

Example: some 3rd party libraries want bytes instead of text, e.g. lxml wants bytes for XML with an encoding declaration

Why not Unicode everywhere for Python 2 and 3?

Gotcha: The Internet is wrong!

- If Python 3's str() and Python 2's unicode() are the same, why not use it? i.e. Python 3 str() or Python 2 unicode(), add "u" prefix, everywhere?
- future recommends doing this, as do many Python porting guides and blogs
- Many internal and external Python 2 APIs (configparser, CherryPy) not setup to work with text world (unicode) strings — blows up horribly!
- Performance with unicode strings on Python 2 slow
- New concept: "native" or "default" string, whatever string type is default on that version of Python = max compatibility with libraries.
- Defensively code to able to string world or byte world data

All Platforms UTF-8 Encoding by Default

Gotcha: with notepad.exe comes great responsibility

- You'll see a lot more encoding from str to bytes and decoding from bytes to str in Python 3
- We've made it so calling encode() and decode() is platform consistent with utf-8, including Windows
- Pre-splunk 8.0 and Python.org Python 3 encode/decode default for Windows would be "ANSI", aka CP-1252. Default for Linux/macOS was "ASCII". Not the same!
- You MIGHT see file encoding errors! You will have to fix them by fixing the files themselves.

Loading an ANSI/cp1252 file under Splunk Python 3

```
with open(“file-I-saved-in-Notepad.txt”) as fp:
    lines = fp.readlines()
```

```
2019-08-30 15:23:54,285 ERROR   [5d69a1fa42104cdcbd0] error:335 -
Traceback (most recent call last):
  …

  File "/opt/splunk/lib/python3.7/shutil.py", line 79, in copyfileobj
    buf = fsrc.read(length)
  File "/opt/splunk/lib/python3.7/codecs.py", line 322, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x93 in position
14045: invalid start byte
```

Numbers changes: float division by default

Python 2:

```
>>> 3/2
1
```

Python 3:

```
>>> 3/2
1.5
>>> 3//2
1
```

Division now returns float. Audit all places division occurs and decide if it's actually float division '/' or int division '//'

Whitespace issues

```
$ ./splunk start
Splunk> The IT Search Engine.
Checking prerequisites…
    Checking http port [8000]: open
    Checking mgmt port [8089]: open
    Checking appserver port [127.0.0.1:8065]: open
    Checking kvstore port [8191]: open
Traceback (most recent call last):
 File "/opt/splunk/lib/python2.7/site-packages/splunk/clilib/cli.py",

line 17, in <module>
    import splunk.clilib.cli_common as comm
 File

"/opt/splunk/lib/python2.7/site-packages/splunk/clilib/cli_common.py",line 528
    cmd = I
        ^
TabError: inconsistent use of tabs and spaces in indentation
```

Don’t mix spaces and tabs in the same file! Just follow PEP8

Porting Mako Templates

- You cannot use any community created tools because the Python is embedded in Mako and HTML
- You should ensure that the Python code is Python 2 and Python 3 compatible
- You'll have to test Python 2 and 3 compatibility using Splunk 7.x for Python 2 and Splunk 8 for Python 3

3.6

Apps Built By Add-on Builder

Developers

splunk> .conf19

Add-On Builder Python 3 Apps

1. Coming soon!
2. Add-On Builder will be able to produce Python 3 apps from your existing projects!

4

Publishing your app

Admins & Developers

splunk> .conf19

What Should I Remember?

Enterprise 8.0 compatibility **requires** Python 3.7 compatibility. Splunk recommends making all scripts dual Python 2/3 compatible **ASAP**.

This will simplify customer upgrades to 8.0.

5 Resources

Admins & Developers

Go find this deck and talk online!

FN1172: Python 2.7 End-of-Life: What it means for your deployment and apps

Python Compatibility For Admins

Learn more

Aditya Tammana
Product Manager | Splunk

What Should I Remember?

1. Documentation is available, including a guide covering helpful Python 2 and 3 code porting topics.
2. User group Slack channel **#python**
3. Splunk answers topic is **python3**
4. Download **Platform Upgrade Readiness App** on Splunkbase

6. Q&A

Aditya Tammana | Product Manager
Cory Burke | Principal Software Engineer
Samat Jain | Senior Software Engineer

splunk> .conf19

.conf19

splunk>

Thank You !

Go to the .conf19 mobile app to

RATE THIS SESSION