POWERING
POTENTIAL

# A Configuration File Primer

## COMMON TYPES OF CONFIG FILES

- INI files
- XML
- JSON
- YAML

# INI FILES (.INI, .CONF, .CFG)

- Historical, informal
- example:

```
; last modified 1 April 2001 by John Doe
[owner]
name = John Doe
organization = Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server = 192.0.2.62
port = 143
file = "payroll.dat“
Key = Value
```

- Seen in:
  → PHP, Git

## XML

- eXtensible Markup Language
- Open standard, related to HTML
- example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<settings>
    <!-- last modified 1 April 2001 by John Doe -->
    <owner>
        <name>John Doe</name>
        <organization>Acme Widgets Inc.</organization>
    </owner>
    <!-- use IP address in case network name resolution is not working-->
    <database server="192.0.2.62" port="143" file="payroll.dat"/>
</settings>
```

- Verbose, complex

# JSON

- JavaScript Object Notation
- example:

```
{
    "owner": {
        "name": "John Doe",
        "organization": "Acme Widgets Inc."
    },
    "database": {
        "server": “192.0.2.62”,
        "port": 143,
        "default": true,
        "file": "payroll.dat"
    }
}
```

- Simple type system, no comments

07461386126 – Definitely be a string, as a number would cut off the first 0

## YAML

- YAML Ain’t Markup Language
  → (formerly, Yet Another Markup Language)
- Superset of JSON
- Spaces, not tabs (set number of spaces, ideally 2)
- example:

```
{
    "owner": {
        "name": "John Doe",
        "organization": "Acme Widgets Inc."
    },
    "database": {
        "server": “192.0.2.62”,
        "port": 143,
        "default": true,
        "file": "payroll.dat"
    }
}
```

```
owner:
  name: John Doe
  organization: Acme Widgets Inc.
database:
  # use IP address in case network name resolution is
not working
  server: "192.0.2.62"
  port: 143
  default: true
  file: payroll.dat
```

## YAML SYNTAX - STRINGS

While YAML allows string values to be specified without quotes, there are a few scenarios that do require quoting your strings:

- Providing a numeric value that we want interpreted as a string instead of a number.
  ```
  storeNumber: "123"
  ```
- Strings containing special characters (:, {, }, [, ], ,, &, *, #, ?, |, -, <, >, =, !, %, @, \).
  ```
  jmespath: "TextView[@text='Submit']"
  ```
- Strings containing escape sequences that need to be parsed as such.
  ```
  streetAddress: "123 No Street\nPhantom City“
  command: “echo ‘hello’”
  ```

- When a string value behaves in ways you didn’t expect, try putting it in double quotes.

# YAML SYNTAX – MULTILINE STRINGS

Literal style, using the | (pipe) character, which preserves end-of-line characters.

```
- script: |
    ls
    npm build
```

Folded style, using the > (greater than) character, which removes end of line characters.

```
- description: >
    This is a longer description which will
    conveniently be transformed by the YAML parser into
    a string with no end of line characters.
```

# YAML SYNTAX – ARRAYS / LISTS

Two options for arrays: (List of elements) (2 space yaml)

```
names:
- “new text” (- as a space + ‘space’)
- “name 1”
- “name 2”
- “name 3”
```

Or:

```
names: [“name 1”, “name 2”, “name 3”]
```

An array of objects:

```
names:
- name: Barney
  age: 36

- name: Betty
  age: 28
```