Testing Jenkins configuration changes

solidify your JCasC, Job DSL and Pipelines usage

Szymon Datko
szymon.datko@corp.ovh.com

Roman Dobosz
roman.dobosz@corp.ovh.com

1st May 2019 (Labour Day)

About us

Szymon Datko

- DevOps & local Bash wizard
- Open Source software lover
- `makepkg`, not war

Roman Dobosz

- Python expert
- `retro/emul maniac`
- `emerge -vaNDu world`

We already talked about Jenkins

`https://www.youtube.com/watch?v=T7rD--ZOYRQ`

⇑ click me! ⇑

OVH

Short recap: what is this mysterious Jenkins thing?

- One of the most popular automation servers.
- Powerful, Open Source, written in Java.
- Easy to start, configure, manage and use.
- Heavily extensible - plenty of plugins available.
- Widely used by the top IT companies!

redhat NOKIA Microsoft OVH rackspace

DELL facebook NASA NETFLIX cloudbees SPACEX

LinkedIn LINEAGE SONY openstack ECLIPSE FOUNDATION

... and many, many more!

Sources: https://wiki.jenkins.io/pages/viewpage.action?pageId=58001258, https://stackshare.io/jenkins.

Short recap: solution for (nearly) all your problems

There are three plug-ins that do come in handy for Jenkins configuration...

Configuration as Code

Job DSL

Job Pipelines

(Jenkinsfiles)

```
pipeline {
  agent docker:'maven:3.3.3'
  stages {
    stage('build') {
      steps {
        sh 'mvn --version'
        sh 'mvn install'
      }
    }
  }
}
```

OVH

How can we test the configuration?

Basically, we can verify two things:

- syntax,
- what it does.

$\sim$ in analogy to:

- unit tests,
- functional tests.

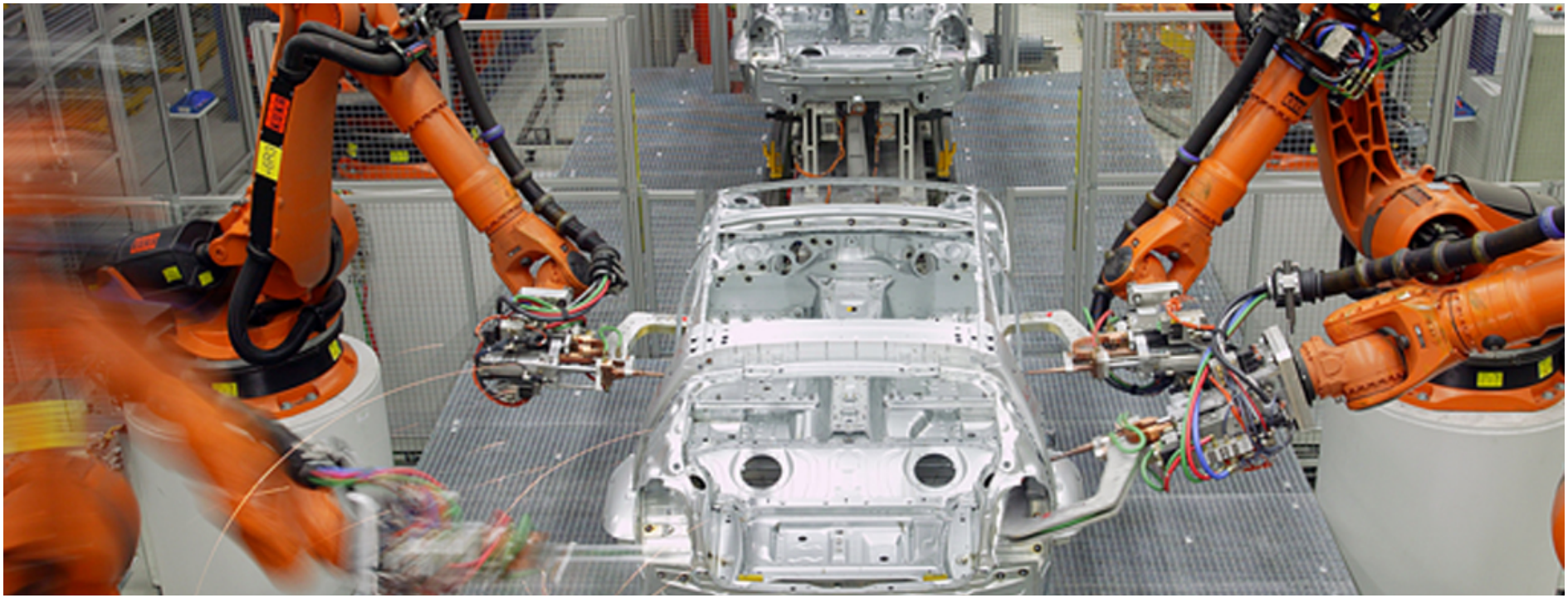
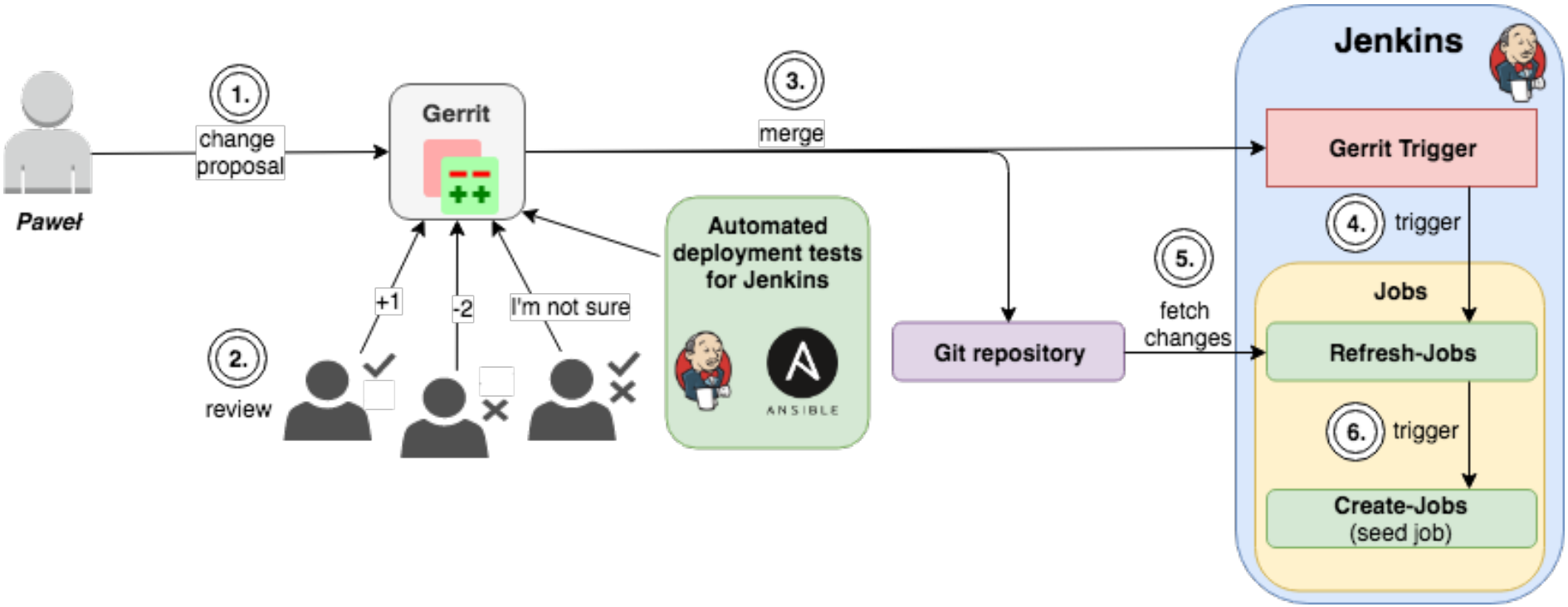


Image source: https://upload.wikimedia.org/wikipedia/commons/c/cf/Application_field_automotive.jpg

OVH

Workflow overview



Verifying Job Pipelines

Simplest to test configuration - parser is built-in in the Pipelines plugin.

Available via:

- Web User Interface,
  - provides hints within textarea field on Pipeline job editing page,
- HTTP API,
  - `${JENKINS_URL}/pipeline-model-converter/validate` endpoint.
  - always sends HTTP 200/OK status code (requires parsing of output),
- SSH CLI,
  - accessible via `declarative-linter` command,
  - requires configured user with `ssh` key and `Overall/Read` permissions,
  - returns nice exit status for shell.

OVH

Verifying Job Pipelines - example script (HTTP)

```
1| #!/bin/bash
2| SEARCH_DIR="${SEARCH_DIR:-.}"
3| JENKINS_URL="${JENKINS_URL:-https://my.jenkins.host.net}"
4| JENKINS_XPATH='concat(//crumbRequestField,":",//crumb)'
5| JENKINS_CRUMB="$(
6|     curl "${JENKINS_URL}/crumbIssuer/api/xml?xpath=${JENKINS_XPATH}"
7| )"
8| errors=()
9|
10| while read -r jenkinsfile_path; do
11|     result=$(curl -X POST -H "${JENKINS_CRUMB}" \
12|                   -F "jenkinsfile=<${jenkinsfile_path}" \
13|                   "${JENKINS_URL}/pipeline-model-converter/validate")
14|
15|     if [ "${result}" != 'Jenkinsfile successfully validated.' ]; then
16|         errors+=($'\n'"FILE: ${jenkinsfile_path}"$'\n'"${result}"$'\n')
17|     fi
18| done < <(find "${SEARCH_DIR}" -iname '*.Jenkinsfile')
19|
20| if [ ${#errors[@]} -gt 0 ]; then
21|     echo 'FAILURE  Syntax errors encountered: THIS SHALL NOT BE MERGED!'
22|     echo "${errors[@]}"
23|     exit 1
24| fi
```

OVH

Verifying Job Pipelines - example script (SSH)

```
1| #!/bin/bash
2| SEARCH_DIR="${SEARCH_DIR:-.}"
3| JENKINS_HOST="${JENKINS_HOST:-my.jenkins.host.net}"
4| JENKINS_PORT="${JENKINS_PORT:-50000}"
5| JENKINS_USER="${JENKINS_USER:-validator}"
6| declare -i errors=0
7|
8| while read -r jenkinsfile_path; do
9|     ssh "${JENKINS_USER}@${JENKINS_HOST}" -p "${JENKINS_PORT}" \
10|         declarative-linter < "${jenkinsfile_path}"
11|
12|     if [ $? -ne 0 ]; then
13|         errors+=1
14|     fi
15| done < <(find "${SEARCH_DIR}" -iname '*.Jenkinsfile')
16|
17| if [ "${errors}" -gt 0 ]; then
18|     echo 'FAILURE  Syntax errors encountered: THIS SHALL NOT BE MERGED!'
19|     exit 1
20| fi
```

If the port number configured is random, one can find it the following way:

```
curl -Lv https://${JENKINS_HOST}/login 2>&1 | grep -i 'x-ssh-endpoint'
```

OVH

Verifying Jenkins Configuration as Code

No reliable syntax checker/linter on the market so far.

However:

- JCasC files are just pure YAML files,
- simple validation can detect obvious syntax errors:

```
python -c 'import yaml,sys; yaml.safe_load(sys.stdin)' < some-file.yaml
```

It is possible to go better:

- JCasC plugin provides the JSON Schema for configuration files under `${JENKINS_URL}/configuration-as-a-code/schema` endpoint,
- it's entries are generated depending on the installed Jenkins plugins,
- it can be utilized for complex validation of JCasC configuration!

OVH

Verifying Jenkins Configuration as Code - validation script

```
1|  #!/usr/bin/env python3
2|
3|  def validate(args):
4|      result = 0
5|      schema = get_schema(args.schema_url)
6|
7|      for fname in args.instances:
8|          with open(fname) as fobj:
9|              try:
10|                 jsonschema.validate(yaml.load(fobj), schema)
11|             except (jsonschema.exceptions.ValidationError,
12|                      jsonschema.exceptions.SchemaError) as err:
13|                 print(err.message)
14|                 result = 1
15|
16|     return result
17|
18|
19| if __name__ == "__main__":
20|     parser = argparse.ArgumentParser()
21|     parser.add_argument('-i', '--instances', nargs='+')
22|     parser.add_argument('-u', '--schema-url')
23|     sys.exit(validate(parser.parse_args()))
```

OVH

Verifying Jenkins Configuration as Code - additonal fixes

```
1| def get_schema(url):
2|     response = requests.get(url, verify=False)
3|     content = response.text
4|
5|     # bad reference
6|     content = content.replace('"type" : "#/definitions/class ',
7|                               '"$ref" : "#/definitions/')
8|     content = content.replace('"type": "#/def', '"$ref" : "#/def')
9|
10|    # remove empty enums
11|    content = re.sub(r',\s*"oneOf" : \[\s*\]', '', content, flags=re.M)
12|
13|    # fix bad names
14|    content = content.replace('[javaposse.jobdsl.dsl.GeneratedItems;',
15|                               'javaposse.jobdsl.dsl.GeneratedItems')
16|    content = content.replace('/[Ljavaposse.jobdsl.dsl.GeneratedItems;"',
17|                               '/javaposse.jobdsl.dsl.GeneratedItems"')
18|
19|    # fix bad references keys
20|    content = content.replace('"ref"', '"$ref"')
21|
22|    schema_dict = json.loads(content)
23|    return schema_dict
```

OVH

Verifying Job DSL

Job DSL is basically an extension of Groovy language;
it might be verified in a programming-like ways.

- syntax parsing:
  - pass a script through the `GroovyShell().parse()`,
  - it may detect basic syntax errors on the script itself,
- unit testing:
  - it is possible to mock script's parts and ensure expected calls are made,
  - this will give you an additional layer of confidence on jobs modification.

OVH

Verifying Job DSL - parsser for DSL scripts

```
1| import groovy.util.CliBuilder
2| import java.io.File
3|
4| class Checker {
5|     static void main(String[] args) {
6|         def cli = new CliBuilder(
7|             usage: 'groovy parse.groovy [groovy-file, ...]'
8|         )
9|         def options = cli.parse(args)
10|
11|         def return_code = 0
12|         for (String fname in options.arguments()) {
13|             File file = new File(options.arguments()[0])
14|             try {
15|                 new GroovyShell().parse(file)
16|             } catch(Exception cfe) {
17|                 System.out.println(cfe.getMessage())
18|                 return_code = 1
19|             }
20|         }
21|         System.exit(return_code)
22|     }
23| }
```

OVH

Is that enough?

Words that seems legitimate does not always have a right meaning.

Taking example of Groovy language, the only way to perform the real validation is to run the code (i.e. as a functional/integration tests).

Image source: https://www.reddit.com/r/funny/comments/2ygq6b/its_is_my_life_jon_bovi/

OVH

Overview of deployment

1. Install Jenkins.
2. Install Jenkins plugins.
3. Add Jenkins configuration.
4. Skip setup wizard.
5. Adjust miscellaneous things.
6. Restart Jenkins.
7. Wait until Jenkins is ready.
8. Trigger seed job.
9. Verify seed job went fine.

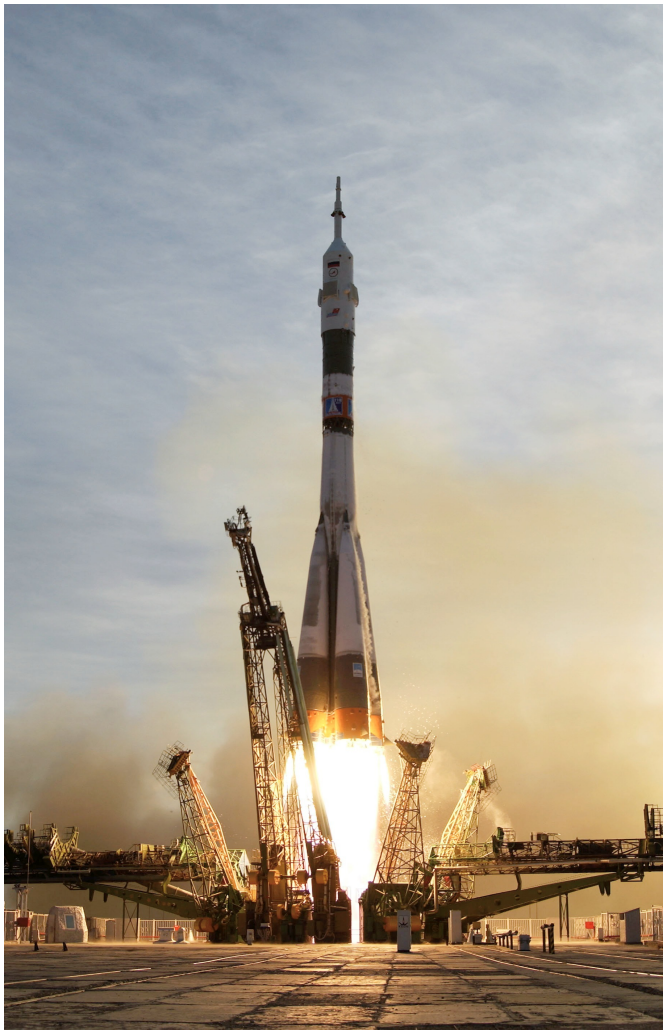


Image source: https://upload.wikimedia.org/wikipedia/commons/f/f4/Soyuz_TMA-5_launch.jpg

OVH

Install Jenkins

You can install it manually, or just use a ready to go Ansible role:

`https://github.com/geerlingguy/ansible-role-jenkins`

```
1| # tasks/install-jenkins.yml
2| ---
3| - name: "Include basic vars"
4|   include_vars: "vars/jenkins_vars.yml"
5|
6| - name: "Install curl"
7|   apt:
8|     name: 'curl'
9|
10| - name: "Install Jenkins"
11|   include_role:
12|     name: geerlingguy.jenkins
```

OVH

Install Jenkins plugins

Simply download Jenkins plugins as `hpi` files and put in `plugins/` directory.

```yaml
# tasks/install-jenkins-plugins.yml
---
- name: "Include plugin list"
  include_vars: "vars/jenkins_plugins.yml"

- name: "Install plugins using specified versions"
  get_url:
    url: "http://updates.jenkins.io/download/plugins/\
          {{ item.name }}/{{ item.version }}/{{ item.name }}.hpi"
    dest: "/var/lib/jenkins/plugins/{{ item.name }}.hpi"
    owner: jenkins
    group: jenkins
    mode: 0644
  register: get_url_result
  until: get_url_result is succeeded
  retries: 3
  delay: 10
  with_items:
    - "{{ jenkins_plugins }}"
```

```yaml
# vars/jenkins_plugins.yml
---
jenkins_plugins:
- { name: "ace-editor", version: "1.1" }
- { name: "ansicolor", version: "0.6.2" }
- { name: "git", version: "3.9.3" }
- { name: "ssh-agent", version: "1.17" }
...
```

OVH

Add Jenkins configuration

```
1  # tasks/configure-jenkins.yaml
2  ---
3  - name: "Create Jenkins Configuration-as-Code directory"
4    file:
5      path: /etc/jcasc
6      state: directory
7      owner: jenkins
8
9  - name: "Fill and copy template based JCasC files"
10   template:
11     src: "files/jcasc/{{ item }}.tpl"
12     dest: "/etc/jcasc/{{ item }}"
13     owner: jenkins
14     mode: 0644
15   with_items:
16     - 'main.yaml'
17     ...
18
19 - name: "Modify Jenkins service file"
20   lineinfile:
21     path: /etc/init.d/jenkins
22     insertafter: "^DAEMON_ARGS="
23     line: "DAEMON_ARGS=\"${DAEMON_ARGS}\
24           --env=CASC_JENKINS_CONFIG=/etc/jcasc/\""
```

We rely on JCasC plugin in this matter completely*

* at least as much as we can...

OVH

Skip setup wizard

Since our configuration is already handled by JCasC plugin, we can omit it.

```
# tasks/skip-setup-wizard.yaml
---
- name: "Insert script to skip Jenkins setup wizard"
  copy:
    src: files/skip-setup-wizard.groovy
    dest: /var/lib/jenkins/init.groovy.d/skip-setup-wizard.groovy
    owner: jenkins
    group: jenkins
    mode: 0644
```

```
#!groovy

import jenkins.model.*
import hudson.util.*
import jenkins.install.*

def instance = Jenkins.getInstance()

instance.setInstallState(InstallState.INITIAL_SETUP_COMPLETED)
```

OVH

Adjust miscellaneous things

This is a right place for:

- installing additional tools, secrets, `ssh` keys and certificates,
- adjusting firewall configuration and setting web server as SSL proxy,
- configuring additional plugins that are not supported by JCasC yet,
  - e.g. locale plugin, Gerrit Trigger.

```
1| # tasks/configure-plugins.yaml
2| ---
3| - name: "Install plugins configuration"
4|   template:
5|     src: "files/config/{{ item }}.tpl"
6|     dest: "/var/lib/jenkins/{{ item }}"
7|     owner: jenkins
8|     group: jenkins
9|     mode: 0644
10|  with_items:
11|    - 'gerrit-trigger.xml'
12|    - 'locale.xml'
```

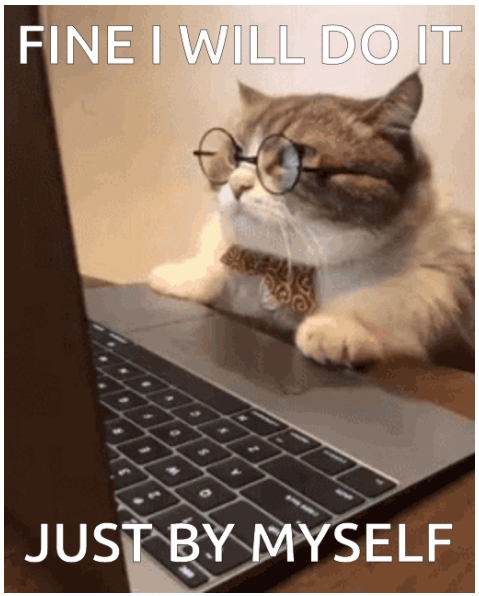


Image source: https://tenor.com/view/business-cat-working-cat-boss-angry-gif-13655998

OVH

Restart Jenkins

Now, once every configuration things are in place, we are ready to re-launch Jenkins with new settings applied.

With `service` module in Ansible it is simple just as that:

```
1| # tasks/restart-jenkins.yaml
2| ---
3| - name: "Restart Jenkins"
4|   service:
5|     name: jenkins
6|     state: restarted
```

Image source: https://me.me/i/friend-where-are-you-me-im-almost-there-7075143

OVH

Wait until Jenkins is ready

During start, Jenkins returns a 503 HTTP response (Service Unavailable).

If it comes up, returning 200 HTTP code, it means JCasC went flawlessly.

Please wait while Jenkins is getting ready to work ...

Your browser will reload automatically when Jenkins is ready.

```
1|  # tasks/wait-for-jenkins.yml
2|  ---
3|  - name: "Wait for Jenkins to come up"
4|    uri:
5|      url: "http://127.0.0.1:{{ jenkins_http_port }}/"
6|      status_code: 200
7|    register: result
8|    until: result.status == 200
9|    retries: 50
10|   delay: 5
```

OVH

Trigger seed job

- Achieved by sending `POST` request to the job's endpoint,
- it may be required to authenticate and/or get CSRF token first,
- also an approval for seed job in Script Security plugin may be needed.

```yaml
# jcasc/security.yml.tpl
---
security:
  globaljobdslsecurityconfiguration:
    useScriptSecurity: False
```

```yaml
# tasks/trigger-seed-job.yaml
---
- name: "Get CSRF crumb for requests"
  uri:
    url: "http://127.0.0.1:{{ jenkins_http_port }}/crumbIssuer/api/json"
    return_content: yes
  register: crumb

- name: "Trigger seed job"
  uri:
    url: "http://127.0.0.1:{{ jenkins_http_port }}/job/Create-jobs/build"
    method: POST
    status_code: 201
    headers:
      'Jenkins-Crumb': "{{ crumb.json.crumb }}"
```

Verify seed job went fine

- As a last step we need to ensure the triggered seed job went fine,
- if it did so, it means the Job DSL created jobs without issues,
- when we succeed here, all our deployment is fine!

```
1|  # tasks/verify-seed-job-success.yml
2|  ---
3|  - name: "Verify seed job result"
4|    uri:
5|      url: "http://127.0.0.1:{{ jenkins_http_port }}\
6|            /job/Create-jobs/lastBuild/api/json"
7|      return_content: yes
8|    register: output
9|    until: output.json.result != 'SUCCESS'
10|   retries: 50
11|   delay: 5
```

Image source: http://highstakesbassin.com/falcon-report/charades-highstakesbassin-com-style/attachment/can/

OVH

Is this enough?

Probably not. What we have achieved so far:

- JCasC configuration validation against its JSON schema,
- jobs definitions syntax (Job DSL) is parsed,
- jobs steps (Job Pipelines) syntax is checked,
- functional test for JCasC as Jenkins deployment job,
- jobs definitions (Job DSL) are also tested during deployment.

What is still missing:

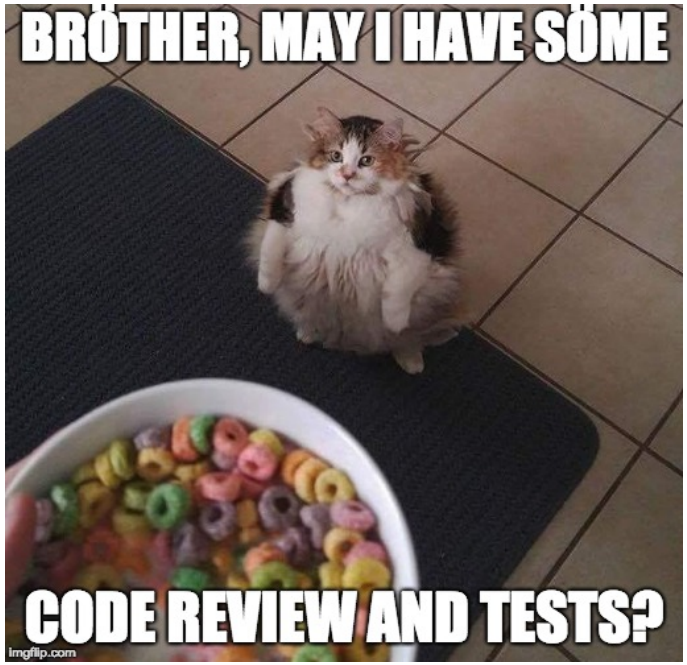
- functional test for Jenkins–Gerrit integration,
- Job Pipelines correctness in isolated environment.

OVH

Conclusions

To sum up:

- testing things is important,
- valid configuration is as important as valid code,
- **Jenkins Configuration as Code**:
  - validate against JSON Schema,
- **Job DSL**:
  - use regular Groovy parser,
- **Job Pipelines**:
  - check with build-in parser,



- additional things need to be launched for completeness!

OVH

Thank you for your attention!

The slides are available: `http://datko.pl/OS-Denver.pdf`

OVH

Testing Jenkins configuration changes

solidify your JCasC, Job DSL and Pipelines usage

Szymon Datko
szymon.datko@corp.ovh.com

Roman Dobosz
roman.dobosz@corp.ovh.com

1st May 2019 (Labour Day)