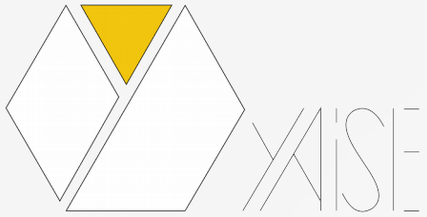


User report:

Jenkins Pipelines for faster CI

C++FRUG #20
2017-11-21 - Murex, Paris, France
Patrick Boettcher YAISE

About me

- Kernel developer since 2004 – (media drivers)
- Embedded C++ and C (most Linux and RTOS)
- Freelancer with YAISE

- this talk is based on work done with VSORA
  - company designing hardware DSP-IPs

Overview

Original problem

Solution with Jenkins

New solutions – new problems

Demo, Examples, Snippets

Problem

- *Continuous Integration* and *Testing*
  - ideally after each commit
  - need a job-server
- Complex arch with *multi-dimensional* build-stages and products and dependencies
- CPU-consuming builds and tests

Problem

- Stage 1: (re)-build tools and toolchains
  - (LLVM/Clang, system-libraries)
- Stage 2: build support-library (3 flavours)
- Stage 3: build platforms (5)
- Stage 4: build test-applications
  - 3 libraries, in Debug and RelWithDebugInfo-mode
- Stage 5: run tests

  ~130 tests per platforms x build-mode x test-configs

Current solution

Jenkins

- Currently done with several Jenkins Jobs (Matrix and FreeStyle)
  - Configured via the web-interface (good starting point)
- Jobs have dependencies to workspaces (no artefacts)
  - jobs are using generated files of others
- One Jenkins master, one slave
  - extremely simple to add slaves (with Linux at least)
- Test-result-analysis, mail sending, reporting

Current solution

Jenkins

- A whole bunch of plugins
- Complex configuration (with hidden/advanced options)
- manually/loosely defined dependencies (don’t forget anything!)
- No remote slaves allowed for building (policy)
- To limit number of jobs to maintain, some builds are done duplicated
- takes 45-50 minutes to complete on two 12-cores machines

New solutions

- build/run-scripts committed to source-tree and run by Jenkins
- multijob – plugin
  - simple to integrate, reduces options to check in web-interface
- Jenkins-DSL (domain-specific-language)
  - transforms/updates a job-config based on a script committed with the sources
- Jenkins Pipelines

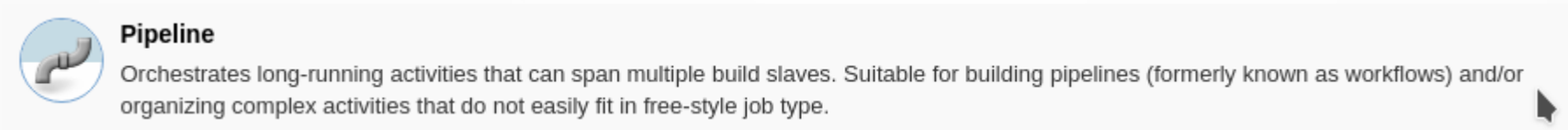
Jenkins Pipeline – Ninja!

“plugins to implement and integrate continuous delivery pipelines into Jenkins.”

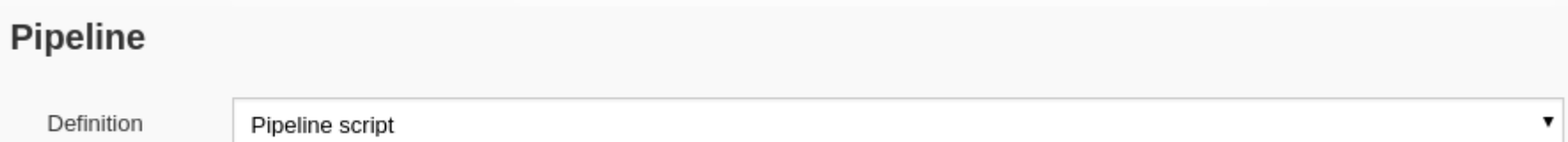
- Test-pipeline is part of the delivery pipeline – fine with me
- “Pipeline-as-Code” - part of the sources – committed and reviewed
- Pipeline-DSL with two syntaxes: Declarative and Scripted
  - I use the scripted one, more possibilities, more complex
- Based on the Apache Groovy-language sugared with Jenkins-objects

Example 1 – Run a job

- Create a pipeline-job



- Add a script




```
node {
    println "update and build toolchain"
    sh 'sleep 5'
}
```


Example 2: Multiple jobs

- node filtered with ‘name’

```
node('master') {
    println "update and build toolchain"
    sh 'sleep 5'
}

node('slave0') {
    println "update and build toolchain"
    sh 'sleep 5'
}
```


Example 3: Parallelization

- Helper functions might need ‘Permissive Script Security Plugin’

```
hosts = hostNames('builder') // hosts with builder-label
jobs = [:]  // maps of jobs

hosts.each { host ->
    jobs["build-on-${host}"] = {
        node(host) {
            println "update and build toolchain"
            sh 'sleep 5'
        }
    }
}

stage("toolchain") { parallel jobs }
```


Example 4: Stash / Unstash

```
types.each { type ->
    jobs["build-${type}"] = {
        node('builder') {
            /* […] */
            stash name: "exe-${type}",
                  includes: "test.sh"
  }}} // bad style but fits on slide ;)

stage("programs") { parallel jobs }

types.each { type ->
    for (i = 0; i < 10; i++) {
        jobs["test-${type}-${i}"] = {
            node('tester') {
                unstash "exe-${type}"
                /* […] */
  }}}}

stage("tests") { parallel jobs }
```

- stages are blocking until all jobs are done

Snippets: ctest + xUnit

- XUnitPublisher can process ctest-result-files

```
node {
  sh "ctest --no-compress-output -T test -R 'filter-regex"

  step([
        $class: 'XUnitPublisher',
        testTimeMargin: '3000',
        thresholdMode: 1,
        thresholds: [],
        tools: [[
                  $class: 'CTestType',
                  deleteOutputFiles: true,
                  failIfNotNew: true,
                  pattern: 'Testing/**/Test.xml',
                  skipNoTestFiles: false,
                  stopProcessingIfError: true]]
       ])
}
```


Snippet: get Shell output

- Call a script, capture the output, use it to generate Jobs

```
node {
    def output = sh script: 'find /bin', returnStdout: true
    output.trim()
    files = output.split("\n")
}

jobs = [:]

files.each() { f ->
    jobs['test-' + f] = {
        node {
            println f
        }
    }
}

stage(“run”) { parallel jobs }
```


Misc.

- Jenkins 2.0 – Blue Ocean
  - new interace based on Pipeline-jobs
- Dynamic slave provisioning (cloud)
  - if your Internet-connection allows it
- Snippet generator
  - Jenkins includes a snippet generator for pipeline
    - http://localhost:8080/pipeline-syntax/
- Good documentation on https://jenkins.io/

Thanks.

p@yai.se

@PatBoeFra

https://github.com/pboettch

https://yai.se/