

# Advanced Jenkins Workshop

Anton Weiss,  
Otomato

# Jenkins CI - the Basics

- Get some code from VCS
  - Setup hooks/polling/scheduling
- Run maven/make/gradle/msbuild/grunt/gulp...
  - Unit tests?
- Deploy build products (Artifactory/S3/network)
- Provision testing environment(s)
- Run integration tests

# Jenkins CI - the Basics

- Manual job creation/editing
- Basic parameters
- Plugins provide most of the functionality
- Statically defined presentation/notifications

# Jenkins CI - beyond the Basics

- Integrate with External Tools
- Aggregate and process build/release data
- Implement smart triggering
- Create dashboards
- Automate job creation/management

# Jenkins Remote API

- Remote API overview.
- XML API
- JSON API
- Automate Jobs creation

# Jenkins Remote API

- What can you do?
  - Retrieve information.
  - Trigger jobs.
  - Copy/Create jobs
- XML, JSON or Python

# Jenkins Remote API

- Authenticating:

- API Token : `http://<jenkins_url>/user/<username>/configure`

- With CSRF Protection enabled:

- Get the crumb:

```
wget -q --auth-no-challenge --http-user <user> --http-password <token> --output-document - 'http://jenkins:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'
```

results in:

```
> Jenkins-Crumb:beae29bc2ca4d5539ba61d1afc410e96
```

now use with curl:

```
curl -H "Jenkins-Crumb:beae29bc2ca4d5539ba61d1afc410e96" -u "user:token" http://jenkins.otomato.link:8080/<endpoint>
```

# Jenkins Remote API

- Submitting Jobs:
  - In job config: Build Triggers -> Trigger builds remotely (e.g., from scripts) -> Authentication Token
  - perform an HTTP POST on JENKINS\_URL/job/JOBNAME/build?token=<TOKEN>

```
curl -X POST -H "Jenkins-Crumb:.." -u "user:token"  
http://jenkins:8080/job/my_job/build?token=TOKEN
```



# Jenkins Remote API

- Submitting Jobs with Parameters:

```
curl -X POST -H "Jenkins-Crumb:..." -u "user:token"  
http://jenkins:8080/job/my_job/build \  
  --data token=TOKEN \  
  --data-urlencode json='{ "parameter":  
[ { "name": "DEBUG_LEVEL", "value": "high" },  
{ "name": "OS", "value": "Ubuntu" } ] }'
```

# Exercise 1:

- Create a job with 2 parameters: STR (type string) & FLAG (type boolean)
- Build step: shell - print both parameters to stdout.
- Trigger the job remotely with values: “API” and “true”
- Question: what happens if we pass “on” as the value for FLAG?

# XML API

- Access data exposed in HTML as XML for machine consumption.
- Use bash, perl, etc.
- Create custom reports and summaries.
- Basic description : <http://your-jenkins:8080/api>
- E.g:
  - <http://jenkins.otomato.link:8080/job/myjob/api/xml> - for job object
  - <http://jenkins.otomato.link:8080/job/myjob/2/api/xml> - for build object

# XML API

- XPathselection with 'xpath':

- `http://jenkins.otomato.link:8080/job/myjob/api/xml?xpath=/*/displayName`

- XPathselection with 'exclude':

- `http://jenkins.otomato.link:8080/job/myjob/api/xml?exclude=/*/build`

- Depth control: (default depth = 0)

- `http://jenkins.otomato.link:8080/api/xml?depth=1`

# Exercise 2

- Retrieve an xml with only the 'fullDisplayName' for 'lastStableBuild' for any job on your jenkins instance.

# XML API

- Limiting the amount of fetched data with 'tree':

- [http://jenkins.otomato.link:8080/view/QA/api/xml?tree=jobs\[name\]](http://jenkins.otomato.link:8080/view/QA/api/xml?tree=jobs[name])

# Exercise 3

- Retrieve an xml with only the 'fullDisplayName' for 'lastStableBuild' for all the jobs on your jenkins instance.

# Python API

```
import requests
url = 'http://jenkins.otomato.link:8080/api/python'
user, password = 'otomato', '3e31e507cea5355de93b28b1fc2d0877'
headers = {'Jenkins-Crumb': 'beae29bc2ca4d5539ba61d1afc410e96'}
r = requests.get(url, headers=headers, auth=(user, password))
r.text
```



# Remote API : Create a Job

- Use an existing jobs config.xml as a template
- Post an edited config.xml to `http://jenkins_url/createItem?name=<JOB_NAME>`
- Add "Content-Type: application/xml" header!

```
curl -X POST -H "Jenkins-Crumb:beae29bc2ca4d5539ba61d1afc410e96" -u  
"otomato:3e31e507cea5355de93b28b1fc2d0877" -H "Content-Type:  
application/xml" http://jenkins.otomato.link:8080/createItem?  
name=new_job1 -d @config.xml
```

# Exercise 4

- Use the config.xml of the job created in Exercise 1 as a template. Add a job description and a 'shell' build step to output current date.
- Create a new job with the name 'date\_job' through remote API.

# Copy a job

- To copy a job, send a POST request to `http://jenkins_url/createItem` with three query parameters:  
`name=NEWJOBNAME&mode=copy&from=FROMJOBNAME`

# Additional Endpoints

- Build queue has its own separate API:

`http://jenkins.otomato.link:8080/queue/api/json (xml/python)`

- Overall load statistics of Jenkins has its own separate API.

`http://jenkins.otomato.link:8080/overallLoad/api/xml`

- Restarting Jenkins
- Enter the "quiet down" mode: ***<http://jenkinsurl/quietDown>***
- Cancel "quiet down" mode: ***<http://jenkinsurl/cancelQuietDown>***
- Restart Jenkins (if installed as a Windows service) : ***<http://jenkinsurl/restart>***
- Restart once no jobs are running: ***<http://jenkinsurl/safeRestart>***

# Grooving with Jenkins

- Using Groovy to:
  - Perform Maintenance Tasks
  - Extend Functionality (instead of writing a full-fledged plugin)
  - Retrieve Information

# The Groovy Language

- Easy to learn and use
- Compiles to JVM bytecode
- Dynamic, optionally-typed
- Fully interoperates with Java
- Provides scripting capabilities
- Allows DSL authoring
- Meta-programming
- Functional programming



# Hello Groovy World!

```
def sayHello(name) {  
    println("Hello $name!")  
}
```

```
def name = 'world'  
sayHello(name)
```

# Groovy Script Console

<http://jenkinsurl/script>

## Script Console

Type in an arbitrary [Groovy script](#) and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use `System.out`, it will go to the server's stdout, which is harder to see.) Example:

```
println(Jenkins.instance.pluginManager.plugins)
```

All the classes from all the plugins are visible. `jenkins.*`, `jenkins.model.*`, `hudson.*`, and `hudson.model.*` are pre-imported.

1

Run



# Groovy Script Console

```
//update git urls for all jobs
import jenkins.plugins.git.*
Jenkins.instance.items.each() { job ->
    def SCM = job.getScm()
    if(SCM instanceof GitSCM ) {
        def url = SCM.userRemoteConfigs[0].url
        println url
        new_url = url.replaceAll(/antweiss/, "otomato_gh")
        SCM.userRemoteConfigs[0].url = new_url
        job.save()
    }
}
```

# Groovy Script Console

```
//reset you job build count
```

```
item = Jenkins.instance.getItemByFullName("your-job-name-  
here")
```

```
//THIS WILL REMOVE ALL BUILD HISTORY
```

```
item.builds.each() { build ->
```

```
    build.delete()
```

```
}
```

```
item.updateNextBuildNumber(1)
```

# Groovy Script Console

- No version control for scripts
- Requires manual editing to change values

● Solution:

- Jenkins Scriptler Plugin

<https://wiki.jenkins-ci.org/display/JENKINS/Scriptler+Plugin>

# Groovy for Dynamic Parameters

- Retrieve parameter values dynamically:
  - from Jenkins itself
  - from external data sources:
    - e.g: SCM, DB, file system
- The script must return a `java.util.List`, an `Array` or a `java.util.Map`, as in the example below:

# Groovy for Dynamic Parameters

- Retrieve a list of successful builds of a project:

```
job = Jenkins.instance.getItem( "myjob" )
def good_builds = []
job.getBuilds().each() { build ->
    if ( build.result.toString() == "SUCCESS" )
    {
        good_builds.add(build.number)
    }
}
//return value is a list
good_builds
```

# Groovy for Dynamic Parameters

- Use with:
  - [Dynamic Parameter Plugin](#) (+Scriptler)
  - [Extended Choice Parameter Plugin](#)
  - [Extensible Choice Parameter Plugin](#)
  - [Active Choices Plugin](#) - the most advanced functionality



# Resources: