Setup Uyuni Development Environment with focus on Java and IntelliJ

Date:	2022-03-09
modified:	2022-05-18
tags:	Uyuni, Java, Development, sumaform, IntelliJ
description:	Start hacking Uyuni and setup your dev environment
category:	Code
slug:	setup_uyuni_development_environment_with_focus_on_java_and_intellij
Author:	Dominik Wombacher
lang:	en
transid:	setup_uyuni_development_environment_with_focus_on_java_and_intellij
Status:	published

Hacking Uyuni, either to troubleshoot and fix bugs or to contribute new features / improvements is something I do since quite a while already, at least whenever I find some time. Getting started wasn't that easy, there are a lot of resources available but sometimes there are outdated, too focused on SUSE employees instead community contributors or just didn't fully answered my questions.

So I want to share my experiences to Setup a Development Environment for Uyuni, with focus on the Java Codebase, on a openSUSE Tumbleweed System by using sumaform and IntelliJ IDEA.

Resources

You can find a lot of information in the Uyuni Wiki and a great Presentation (Archive, Source) from Cédric Bosdonnat.

sumaform

See my Post Uyuni Test Environment with sumaform on local libvirt host (openSUSE Tumbleweed)

There are Alternative Instructions (Archive: [1], [2]) available in case using sumaform isn't possible or not intended.

Install Packages

Add the *systemsmanagement:Uyuni:Utils* repository, especially for *obs-to-maven*, and install the necessary packages.

Manually:

```
sudo zypper addrepo obs://systemsmanagement:Uyuni:Utils systemsmanagement:uyuni:utils
sudo zypper in java-11-openjdk-devel openssh rsync apache-ivy ant ant-junit servletapi5 cpio obs-to-maven tito yarn
```

Ansible snippet:

```
- name: Import systemsmanagement:/Uyuni:/Utils RPM Key
  rpm_key:
    key: https://download.opensuse.org/repositories/systemsmanagement:/Uyuni:/Utils/openSUSE_Tumbleweed/repodata/repomd.xml.key
    state: present

- name: Add systemsmanagement:/Uyuni:/Utils RPM Repository
  community.general.zypper_repository:
    name: systemsmanagement_uyuni_utils
    description: Several utilities to develop, build or release Uyuni (openSUSE_Tumbleweed)
    repo: https://download.opensuse.org/repositories/systemsmanagement:/Uyuni:/Utils/openSUSE_Tumbleweed/
    priority: 95
    state: present
  ignore_errors: true

- name: Package Installation (Uyuni Development)
  community.general.zypper:
    name:
      - java-11-openjdk-devel
      - openssh
      - rsync
      - apache-ivy
      - ant
      - ant-junit
      - servletapi5
      - obs-to-maven
      - tito
      - yarn
    allow_vendor_change: true
    force_resolution: true
    force: true
    state: latest
```

Clone git repo and prepare files / folders

Fork the Uyuni Repository and clone it. I will use the placeholder `<path_to_uyuni>` a lot, which refers to the local path of your cloned uyuni fork.

Let's start with some files and folders which are required at a later point for unittest and deployment.

Create the folders */usr/share/rhn/config-defaults*, */var/log/rhn* and */srv/susemanager*, owner and group should match with your user account.

Ansible snippet:

```
- name: Create Folders for Uyuni Development Unittests and Build
  file:
    path: "{{ item }}"
    state: directory
    owner: wombelix
    group: users
  loop:
    - /usr/share/rhn/config-defaults
```

```
    - /var/log/rhn
    - /srv/susemanager
```

Create a *rhn.conf* used by JUnit:

```
cp <path_to_uyuni_root>/java/buildconf/test/rhn.conf.postgresql-example <path_to_uyuni_root>/java/buildconf/test/rhn.conf
```

Get / update java libraries and dependencies:

```
cd <path_to_uyuni_root>/java
ant -f manager-build.xml ivy
```

Compile branding jar for the first time:

```
cd <path_to_uyuni_root>/java
ant -f manager-build.xml refresh-branding-jar
```

Configure IntelliJ IDEA

I will focus on IntelliJ IDEA but can also use Eclipse (Archive: [1], [2]) or VSCode (Archive: [1], [2]), if you want.

The following Steps are heavily based on IntelliJ IDEA specific development instructions (Archive: [1], [2]) and Java Development Environment (Archive: [1], [2]) with some adjustments, additional information and tested on IntelliJ IDEA Ultimate 2022.1 EAP.

The JetBrain Toolbox is in my opinion the easiest way to install and update IntelliJ. If you installed it manually, please check the documentation where / how you can configure the *vmoptions*.

Toolbox App > three dots next to "IntelliJ IDEA" > Settings > Configuration > Java Virtual Machine options "Edit..."

> Replace `-Xmx2048m` with `-Xmx4G`

Source: https://intellij-support.jetbrains.com (Archive: [1], [2])

Afterwards start IntelliJ IDEA to proceed with the actual configuration.

File > New > Project from existing Source

> Select <path_to_uyuni_root>
> Create Project from existing Source
> Accept the project format defaults
> Also accept auto discovered source directories
> From the Libraries list, uncheck all items
> From the Modules list, only check the items corresponding to the following directories:
>
> > `uyuni/java/code` (change the name to code)
> > `uyuni/branding/java/code` (change the name to branding)
>
> Select a Java 11 runtime e.g. the previously installed openJDK
> Ultimate Edition: Unselect eventually found frameworks

Enable automatic building

File > Settings... > Build, Execution, Deployment > Compiler and select "Build project automatically"

Configure Code Style

File > Settings... > Editor > Code Style > Java > Imports

Click on the cog / settings icon next to the "Scheme: Default" field at the top, then "Import Scheme", "IntelliJ IDEA code style XML" and select the `<path_to_uyuni_root>/java/conf/intellij-codestyle.xml` file.

Remote Debugging

Run > Edit Configurations... > + sign

Accept all defaults, except from Host and Port, configure them based on the service you want to debug. 8000 for Tomcat, 8001 for Taskomatic, 8002 for Search (defaults if deployed with sumaform)

Further reading: IntelliJ IDEA Debugging Guide (Archive: [1], [2])

Ivy integration

File > Settings... > Plugins > Browse repositories... > search for "IvyIDEA" > Install the Ivy plugin

Restart IntelliJ IDEA (if asked) to activate the plugin

File > Project Structure... > Modules -> right click on code > + sign > click on "ivyIDEA" to enable the plugin for the project

Click on folder icon at the right side to select the Ivy configuration path: `<path_to_uyuni_root>/java/buildconf/ivy/ivy-suse.xml`
Check "Use module specific ivy settings"
Click on folder icon at the right side to select the Ivy configuration path: `<path_to_uyuni_root>/java/buildconf/ivy/ivyconf.xml`

Tools > IvyIDEA > "Resolve for all modules" to get updated Ivy dependencies

Note: When switching branches that have different dependencies (notably, major versions) you have to:

Tools > IvyIDEA > Remove all resolved libraries
Tools > IvyIDEA > Resolve for all modules
Build > Rebuild project

CheckStyle integration

File > Settings... > Plugins > Browse repositories... > search for "CheckStyle" -> Install the CheckStyle IDEA plugin

Restart IntelliJ IDEA to activate the plugin

File > Settings... > Tools > Checkstyle

Change the Checkstyle version to the one in `<path_to_uyuni_root>/java/buildconf/ivy/ivy-suse.xml` (currently 8.30)
Click on the + sign next to Configuration File

Description: Uyuni
Check "Use a local Checkstyle file", select `<path_to_uyuni_root>/java/buildconf/checkstyle.xml`
Check "Store relative to project location", click on Next

Set the following values for properties:

```
checkstyle.cache.file: <path_to_uyuni_root>/java/build/checkstyle.cache.src
checkstyle.header.file: <path_to_uyuni_root>/java/buildconf/LICENSE.txt
checkstyle.suppressions.file: <path_to_uyuni_root>/java/buildconf/checkstyle-suppressions.xml
javadoc.lazy: false
javadoc.method.scope: public
javadoc.type.scope: package
javadoc.var.scope: package
```

Click on Finish, mark the file as Active, click on Apply and leave the Settings.

Afterwards a new mini-tab will appear at the bottom named "CheckStyle".

Avoid CheckStyle violations

These are recommended settings, which might already be set as default, that help respecting style guidelines independent of the CheckStyle plugin:

enabling automatic import completion

File > Settings... > Editor > General > Auto Import

Set "Insert imports on paste" to "Always"
Select "Add unambiguous imports on the fly" and "Optimize imports on the fly" in the Java Section.

disabling "star imports"

File > Settings... > Editor > Code Style > Java > Imports

Class count to use import with '*' > 999
Names count to use static import with '*' > 999

wrapping and braces

File > Settings... > Editor > Code Style > Java > Wrapping and Braces

Under 'try' statement check 'catch' on new line and 'finally' on new line
Under 'if' statement check 'else' on new line

Faster deployments via manager-build.xml

Change the output directory to enable quick *manager-build.xml* deploys:

File -> Project Structure... -> Modules -> code -> Paths

Click on "Use module compile output path" and set:
"Output path" to `<path_to_uyuni_root>/java/build/classes`
"Test output path" to `<path_to_uyuni_root>/java/build/tests`

Enable usage of precompiled files by adding `precompiled=true` to `<path_to_uyuni_root>/java/buildconf/manager-developer-build.properties`, if the file not

exist, copy `<path_to_uyuni_root>/java/buildconf/manager-developer-build.properties.example`, rename and edit the new file.

Configure JUnit tests

File > Project Structure... > Modules > code > Dependencies

Click the + sign > Module dependency > branding > OK to include branding classes and files in the build

File > Project Structure... -> Modules -> code

Mark the directory webapp as Resources

Run > Edit Configurations... > + sign > JUnit

Name: JUnit
Run on: Local machine
Build and run:

```
JRE: Java 11
-cp: -cp code
-ea: -ea -Drhn.config.dir=$MODULE_DIR$/../buildconf/test/ -Dlog4j.threshold=debug
```

Select "All in package" to execute all available Unittests, if you want to limit to a specific class or package adjust the dropdown and filepath accordingly

To start the JUnit tests, click on **Run > Run**.

Important: Start the test database docker container first, otherwise almost all tests will just fail

```
cd <path_to_uyuni_root>/java
make -f Makefile.docker dockerrun_pg
```

Deploying Java code or CSS

If you created a terraform based VM with sumaform, you can easily deploy code:

1. Run checkstyle

```
cd <path_to_uyuni_root>/java
ant -f manager-build.xml checkstyle
```

2. Deploy

```
cd <path_to_uyuni_root>/java
ant -f manager-build.xml refresh-branding-jar deploy -Ddeploy.host=uyuni-server.tf.local restart-tomcat restart-taskomatic
```

You can configure the *deploy.host* in `<path_to_uyuni_root>/java/buildconf/manager-developer-build.properties` and omit the command line parameter.

Contribute

Uyuni exist since July 2018, the initial release (4.0.0) was based on SUSE Manager 3.2, since then Uyuni is the Upstream Project of SUSE Manager. SUSE Manager is based on Spacewalk, which was sponsored by Red Hat and abandoned, so SUSE decided to start a own Fork (Archive: [1], [2]), Uyuni was born.

Still a lot of development comes from SUSE but there is a growing Community (Archive: [1], [2]) with more and more independent Contributions.

If your dev environment is ready and you want to jump in, but didn't contributed to the Uyuni Project before, I suggest you take a look at some Good first Issues.