**Guia de Instalação e Configuração do**

**BDD - MV Lojas**

**Responsáveis:**

Marlon Lopes Pinto

Weverton Castro

**Empresa:** Softbox

**Cliente:** Máquina de Vendas

**Projeto(s):** MV Lojas, BDD

**Contato:**

[marlon@softbox.com.br](mailto:marlon@softbox.com.br)

[weverton@softbox.com.br](mailto:weverton@softbox.com.br)

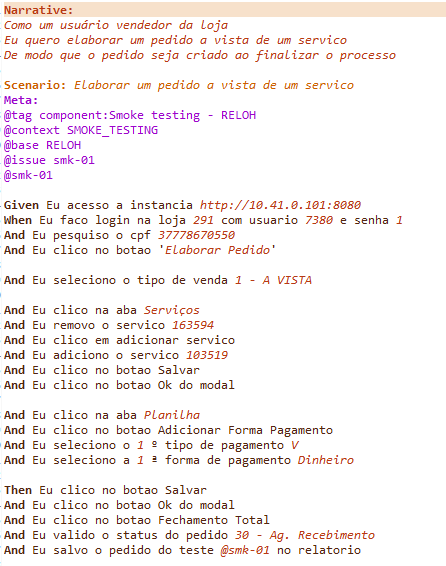
# O que é BDD?

Antes de começar a instalação e configuração do projeto é importante saber o que é o BDD, qual seu propósito e como funciona.

BDD é uma metodologia de desenvolvimento ágil que visa integrar regras de negócios com linguagem de programação, focando no comportamento do software.

O objetivo do BDD é possibilitar que os Analistas possam escrever seus cenários de teste no formato de histórias, especificando os requisitos através de um passo-a-passo.

Segue abaixo o padrão utilizado pela equipe de testes:



**Figura 1:** Cenário de elaboração de um pedido a vista de um serviço.

# Configuração do Ambiente

A princípio a configuração será focada no sistema operacional Windows, abaixo seguem os pré-requisitos necessários.

## Pré-Requisitos

1. Baixar e instalar o [Java](#_72gke3ng7ehr), o mesmo pode ser obtido através do link: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

2. Baixar e instalar o [Eclipse](#_72gke3ng7ehr), o mesmo pode ser obtido através do link:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2>

Configure no Eclipse, o local da instalação do [JDK](#_72gke3ng7ehr) do Java.

* Abra o Eclipse e acesse: Windows > Preferences;
* Acesse a opção Java > Installed JREs;
* Clique no botão “Add”, selecione a opção “Standard VM” e clique no botão “Next”;
* No campo JRE home: informe o caminho da instalação do JDK da sua máquina; Ex.: C:\Program Files\Java\jdk1.8.0\_101 e clique no botão “Finish”;

Para facilitar a escrita das histórias é necessário realizar a instalação do plugin do Jbehave no Eclipse, portanto, siga os passos abaixo:

* Abra o Eclipse e acesse o menu: Help > Install New Software…
* Adicione um novo site: <http://jbehave.org/reference/eclipse/updates/>
* Selecione o plugin JBehave e clique Next, Next e Finish.

Outro plugin necessário é do [Maven](#_72gke3ng7ehr) (m2eclipse):

* Abra o Eclipse e acesse: Help > Eclipse Marketplace...
* Pesquise por: m2eclipse
* Instale o plugin: Maven Integration for Eclipse WTP (Juno)
* Confirme a seleção de todos os recursos do plugin e clique em Next;
* Aceite os termos de licença e clique em Finish;
* Após a conclusão da instalação, reinicie o Eclipse;

3. Baixar e instalar o [Git](#_72gke3ng7ehr), para clonar o repositório do projeto *mv-lojas-bdd* que será utilizado para desenvolvimento dos cenários.

<https://git-scm.com/download/win>

Após o passo acima, devemos configurar uma chave ssh, para tal, precisamos acessar o Gitlab da softbox(<https://gitlab.softbox.com.br>), efetuar login através do usuário de rede.

Acessar o menu do Usuário > Profile Settings > SSH Keys;

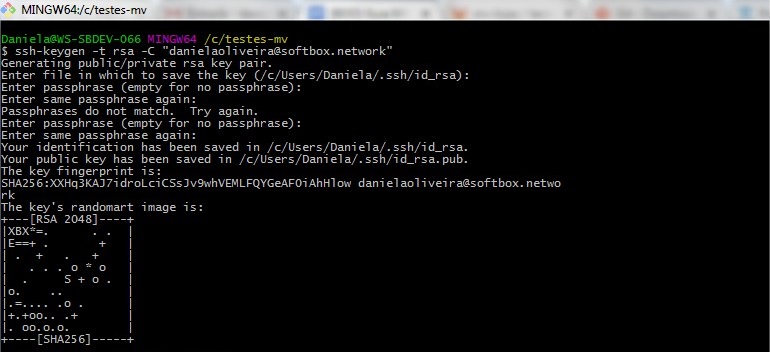
Se você ainda não tiver uma chave gerada, clique no link ([generate it](https://gitlab.softbox.com.br/help/ssh/README)) para pegar seu usuário de rede;

Screenshot_1.png

Abra o programa Git Bash e execute os comandos:

* *ssh-keygen -t rsa -C "SEU USUÁRIO DE REDE"*

(Para gerar sua chave. Obs.: não informe senha)



* *cat ~/.ssh/id\_rsa.pub*

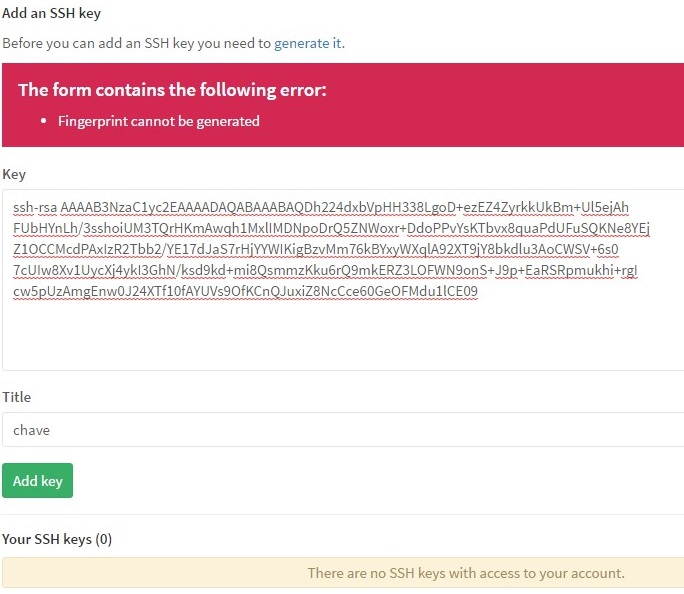
(Para copiar sua chave para a área de transferência. Obs.: Todo o conteúdo do arquivo faz parte da chave)



Após isso, volte no GitLab da softbox e adicione sua chave:

No campo “**Key**”, cole a chave acima, e no campo “**Title**”, informe um título qualquer e que no botão “Add key”.

Caso ocorra o erro abaixo:



Observe se todo o conteúdo do arquivo (*~/.ssh/id\_rsa.pub*) foi copiado para o campo key sem espaçamento.

## Baixando o projeto e executando os testes

Após ter verificado os pré-requisitos siga os passos abaixo:

1. Para padronizar o local onde os arquivos doc projeto serão clonados, crie a pasta *C:\testes-mv*
2. Abra o programa Git Bash e execute o comando seguinte na pasta acima, para baixar o projeto: **$ git clone ssh:**[**//git@gitlab.softbox.com.br**](about:blank)**:1912/mv-lojas/testes-automatizados.git**
3. Importe o projeto no eclipse da seguinte forma:

* Acesse o menu: File > Import > Maven > Existing Maven Projects;
* Busque a pasta onde foi realizado o clone: Ex.: C:\testes-mv\testes-automatizados\mv-lojas-bdd, selecione projeto e Finish;

1. Para fazer o build (execução dos testes) do projeto no eclipse siga os passos abaixo:

* Botão direito no projeto "mv-lojas-bdd", Run As > Maven build…
* No campo "Goals" informe: clean verify

OBS.: Caso queira filtrar por categoria/tags (SMOKE\_TESTING / REGRESSAO), seguem abaixo alguns exemplos;

Para executar somente os testes que contém as tags: @issue smk-05, smk-03, smk-09 e a tag. @base RELOH

* + *clean verify -Dmetafilter="groovy: (issue == 'smk-05' || issue == 'smk-03' || issue == 'smk-09') && (base != 'MVSH')"*

Para executar todos os testes que tenham a tag @context SMOKE\_TESTING e a tag. @base RELOH

* + *clean verify -Dmetafilter="groovy: (context == 'SMOKE\_TESTING' && base != 'MVSH')"*

Para executar todos os testes que tenham a tag @context REGRESSAO

* + *clean verify -Dmetafilter="+context REGRESSAO"*

Para executar somente o teste que contém a tag @rgs-20

* + *clean verify -Dmetafilter="+rgs-20"*

Utilizar o parâmetro -Dmaven.clean.failOnError=false, caso aconteça falha ao executar o passo "clean".

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ mv-lojas-bdd ---

[INFO] Deleting C:\testes-mv\testes-automatizados\mv-lojas-bdd\target

* clean verify -Dmetafilter="+rgs-20"
* -Dmaven.clean.failOnError=false

Executa os cenários de reversão:

* clean verify -Dmetafilter="groovy: (issue == 'rgs-59' || issue == 'rgs-60' || issue == 'rgs-61' || issue == 'rgs-62' || issue == 'rgs-63' || issue == 'rgs-64' || issue == 'rgs-65' || issue == 'rgs-73') && (base != 'MVSH')"

Executa cenários avulsos, 4 pedidos a vista/dinheiro, 4 pedidos cartão de crédito:

* clean verify -Dmetafilter="groovy: (issue == 'avl-03' || issue == 'avl-04' || issue == 'avl-05' || issue == 'avl-06' || issue == 'avl-02' || issue == 'avl-07' || issue == 'avl-08' || issue == 'avl-09') && (base != 'MVSH')"

Executa cenários de reversão em AMBIENTE DE VALIDAÇÃO

* clean verify -Dmetafilter="groovy: (issue == 'rgs-66' || issue == 'rgs-67' || issue == 'rgs-68' || issue == 'rgs-69' || issue == 'rgs-70' || issue == 'rgs-71' || issue == 'rgs-72' || issue == 'rgs-82') && (base != 'MVSH')"

Aba JRE, marque a opção "Alternate JRE": escolha um jdk, Exemplo: jdk1.8.0\_101

## Criando uma branch e desenvolvendo novos cenários BDD

* Atualize a branch master:
  + *git pull origin master*
* Crie sua branch para o desenvolvimento do seu cenário:
  + *git checkout -b nome-da-sua-branch*

Para cada alteração/desenvolvimento no projeto devem ser executados os seguintes comandos:

* Para adicionar todos os arquivos:
  + *git add .*
* ou para cada arquivo:
  + *git add caminho-do-arquivo*
* Após executar um dos dois comandos acima, deve ser feito o commit e adicionar um comentário:
  + *git commit -m 'Mensagem relacionada à alteração'*
* Realizar o envio dos arquivos para o repositório:
  + *git push origin nome-da-sua-branch*

Após a finalização de todo o desenvolvimento do cenário em questão, executar os seguintes comandos para realizar o merge da sua branch com a master:

* *git checkout master*
* *git pull origin master*
* *git merge nome-da-sua-branch*
* *git push origin master*

# Cenários e Histórias BDD

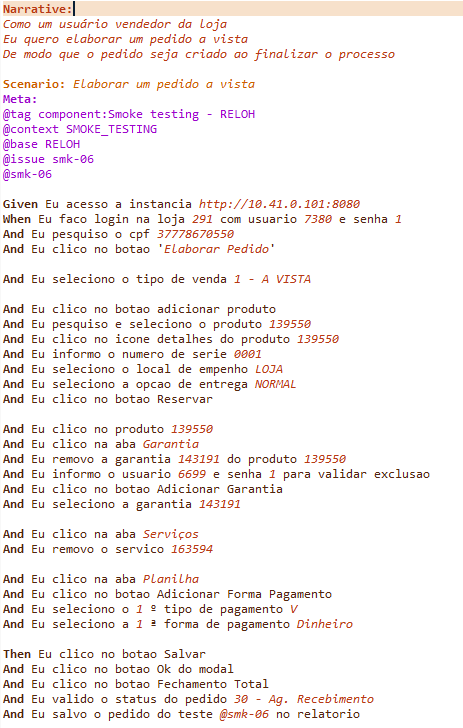
De acordo com a estrutura do projeto *mv-lojas-bdd*, para criar um novo cenário precisamos primeiro conhecer que dentro do projeto existe a pasta */mv-lojas-bdd/src/test/resources/****stories****,* responsável por armazenar as histórias.

Dentro dessa pasta **stories** possuímos outras pastas que definem o tipo ou agrupamento de uma história, como por exemplo: **pedido** ou **processos**;

A partir disso podemos criar um arquivo de uma história específica, Ex.: **pedido\_a\_vista.story** que fica dentro da pasta **pedido.**

Segue abaixo um modelo de como devemos estruturar o conteúdo do arquivo acima.

...

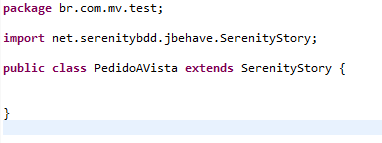


**Figura 2:** Conteúdo do cenário **pedido\_a\_vista.story**

Após criarmos o arquivo **.story**, devemos criar um outro arquivo (**.java**) para que o projeto consiga encontrar essa história e executá-la.

Esse outro arquivo deve ser criado na pasta: /mv-lojas-bdd/src/test/java/br/com/mv/**test**, com o nome referente à história, como por exemplo: **PedidoAVista.java**

Segue abaixo um modelo do conteúdo do arquivo acima.



Após a criação desses dois arquivos, podemos realizar a execução do cenário.

**Obs.:** Caso tenha sido criada uma história que contenha passos que ainda não foram desenvolvidos em java, deve ser solicitado ao desenvolvedor da automação, para que esses passos sejam implementados.

O caminho do relatório do Serenity é **/testes-automatizados/mv-lojas-bdd/target/site/index.html**

# Executando via Docker

Executar o comando para criação de dockers ansible\ansible-playbook init.yml:

Descomentar a linha no arquivo serenity.properties

'#webdriver.remote.url=http://localhost:4446/wd/hub

Executar via gradle

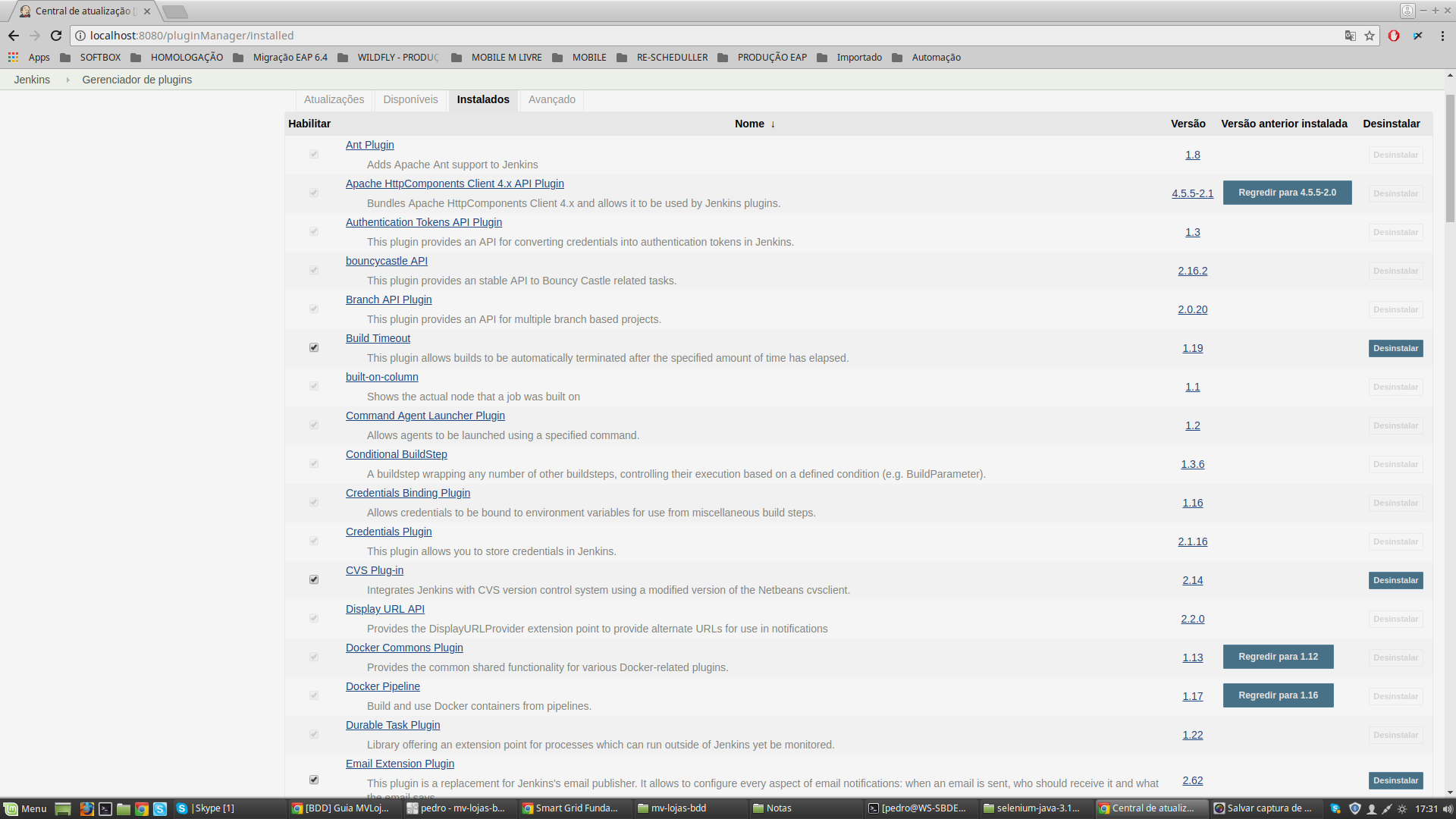
gradle clean build test aggregate -Dmetafilter="+prd-smk-02" -Dmvlojas.urlProducao="http://10.30.232.18:8080" --stacktrace

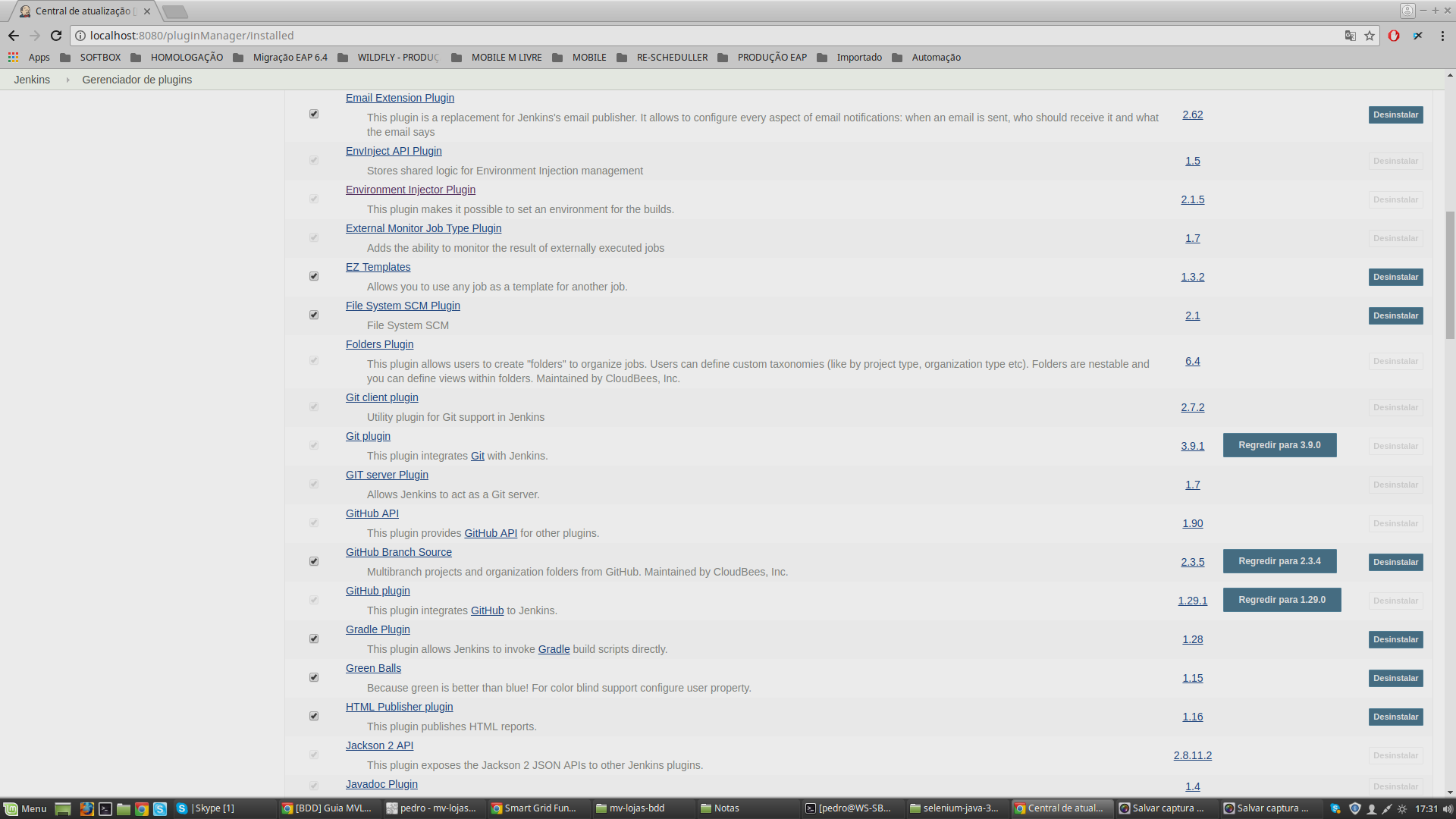
# Jenkins

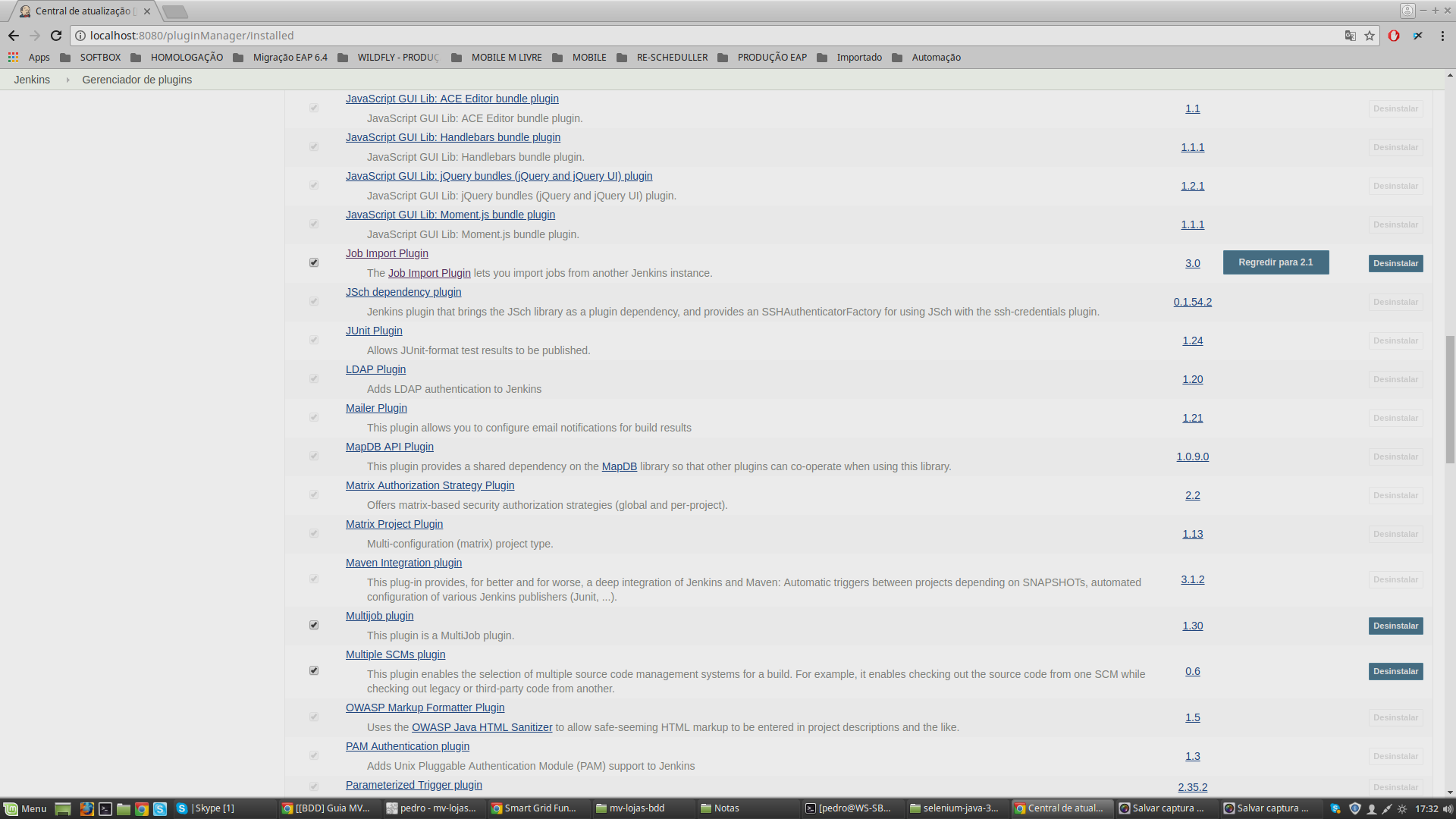
Jenkins é um servidor *opensource* voltado para automação que possui centenas de plugins, facilitando assim o *building*, *deploying* e a automação de qualquer projeto.

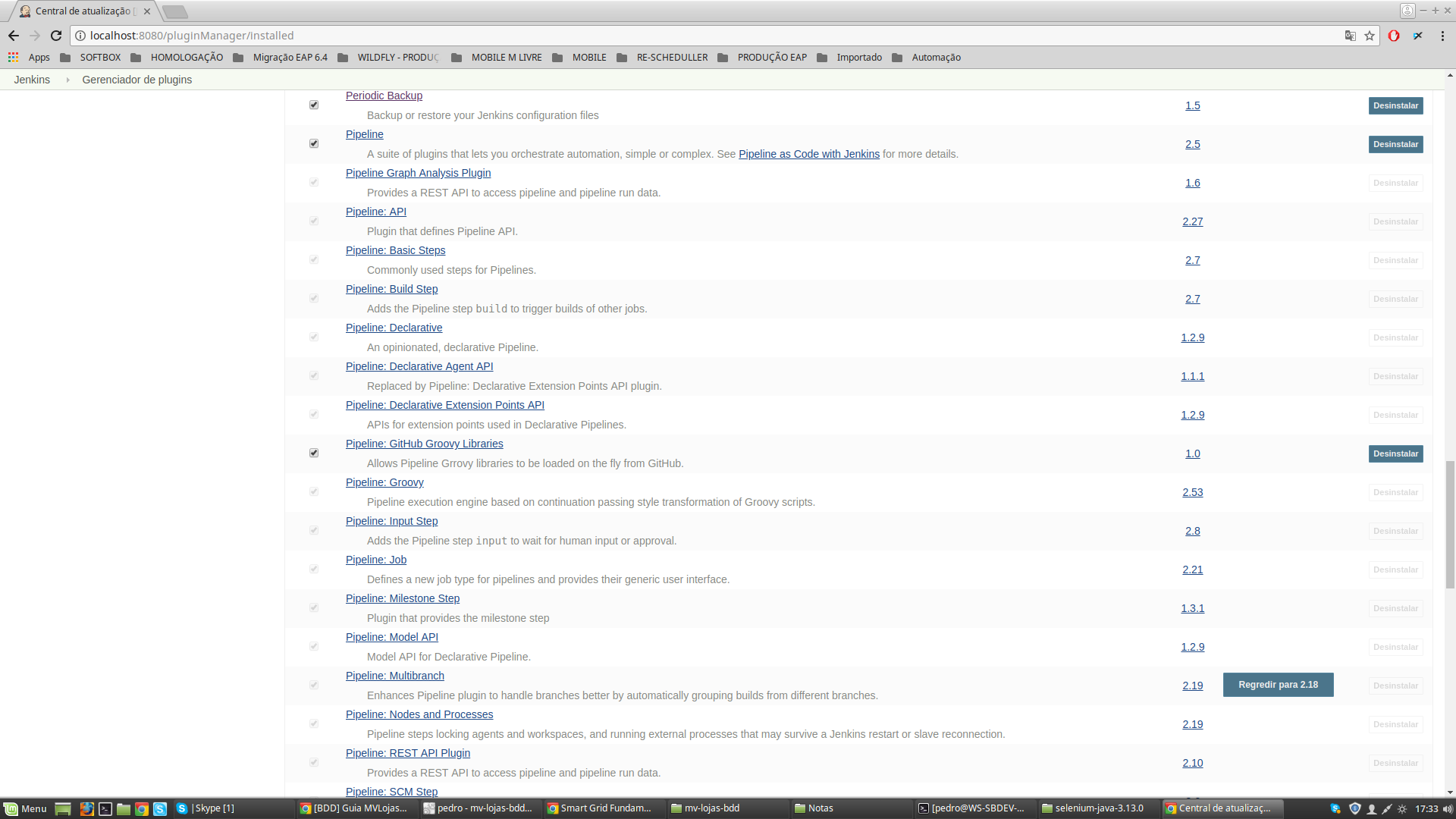
Para baixá-lo, acesse <https://jenkins.io/download/>. Qualquer dúvida na instalação, acesse <https://www.tutorialspoint.com/jenkins/jenkins_installation.htm>

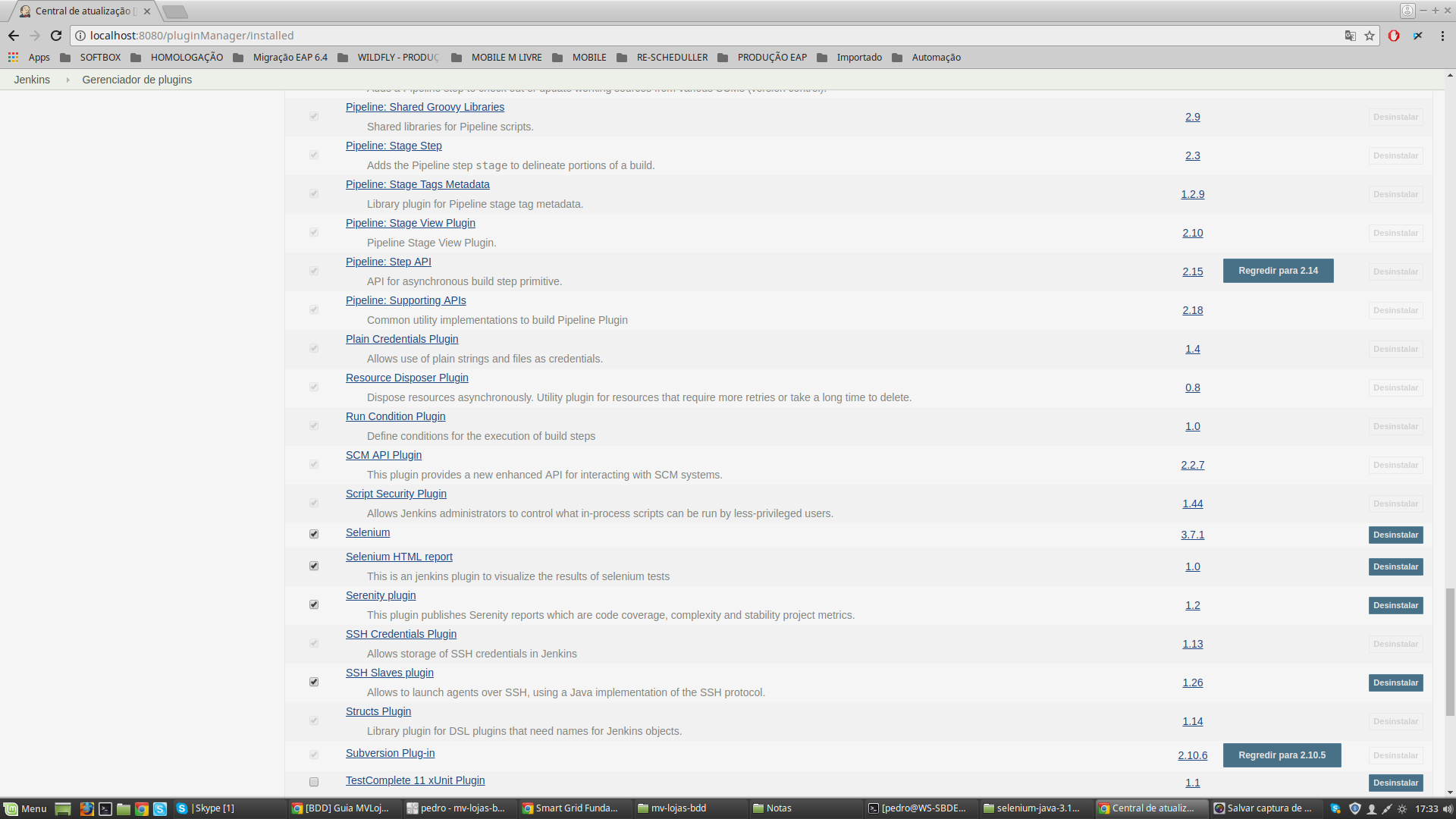
No quesito de configuração, deve-se baixar os plugins necessários (cuja lista se encontra abaixo): acesse o servidor pelo endereço (o padrão é localhost:8080), vá em Gerenciar Jenkins, Gerenciar Plugins, clique na aba Disponíveis e marque a checkbox dos seguintes plugins:

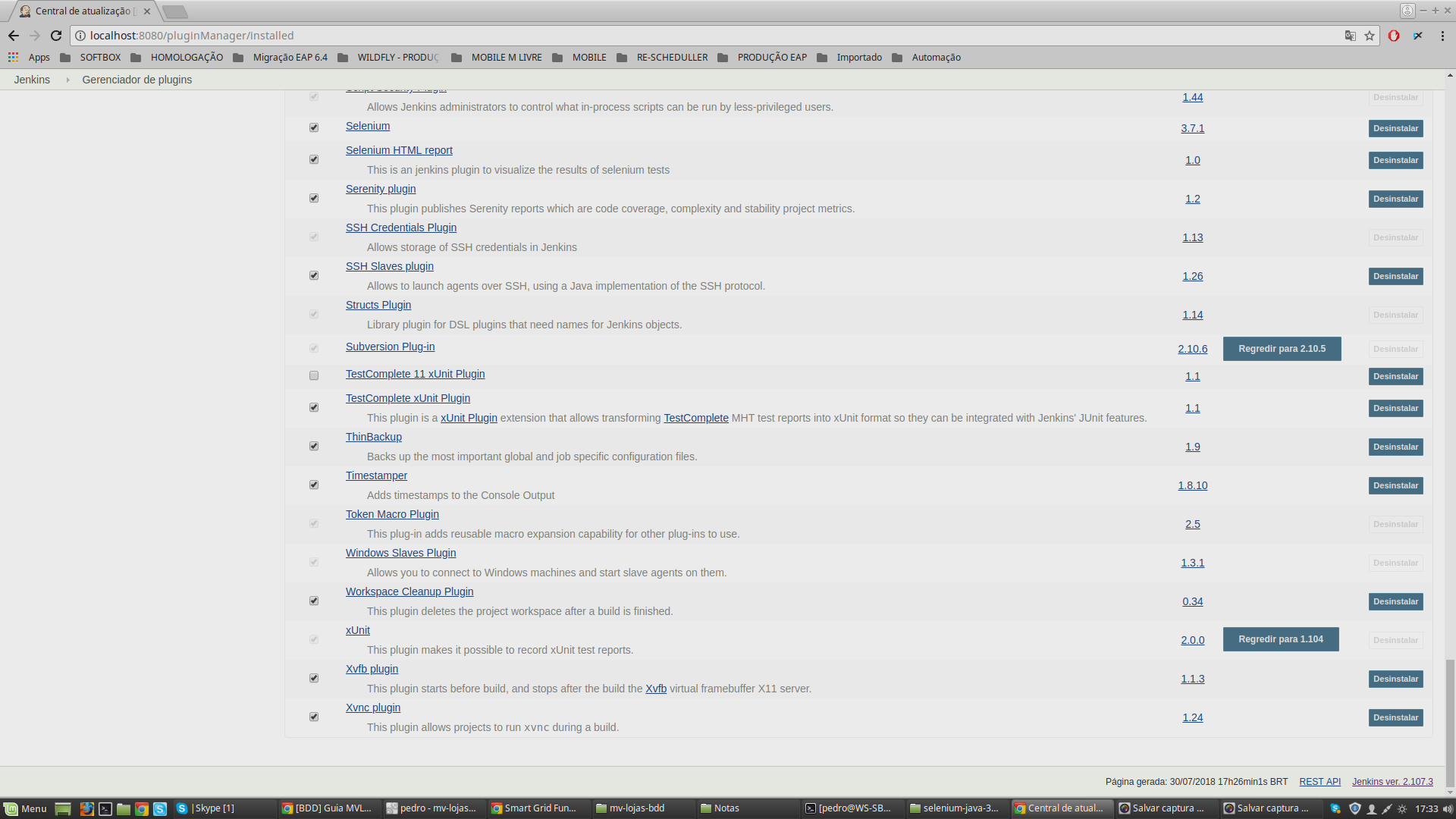












Após isso, é necessário importar os Jobs do já criados. Tendo o arquivo dos Jobs, basta copiá-los para dentro da pasta de jobs, e reiniciar o computador. Caso exista jobs com caracteres especiais, favor renomeá-los de forma ao nome possuir apenas caracteres simples.

# Observações

O Selenium WebDriver é a interface que permite “conduzir” um navegador de forma nativa

assim como um usuário faria, seja na máquina local ou remotamente. Suas bibliotecas devem ser atualizadas no projeto sempre que possível. Para isso:

Baixe a última versão do selenium no link <https://www.seleniumhq.org/download/> (baixar para linguagem Java). Em seguida, abra as Propriedades do projeto vigente no eclipse e vá para **Java Build Path**. Acesse a aba **Libraries** e clique em **Add External JARs**. Selecione a pasta onde se extraiu o arquivo, e adicione todos os arquivos **.jar**, os do selenium e dos dentro da pasta  **lib**.

Obs.: sempre verificar compatibilidade <https://www.seleniumhq.org/about/platforms.jsp>

O Gecko Driver é um proxy para fazer a interação entre clientes compatíveis com W3C WebDriver e navegadores baseados em Gecko, como é o caso do Mozilla FireFox.

No pŕojeto, deve-se indicar o caminho onde se encontra o executável do drive atribuindo-o à variável **webdriver.gecko.driver**, que se encontra em **serenity.properties**.

Recomenda-se atualizar o driver sempre que possível. Para isso, basta acessar <https://github.com/mozilla/geckodriver/releases> e baixar o arquivo mais atual adequado ao sistema operacional.

O ChromeDriver é um servidor autônomo que implementa o protocolo **wire** para o **Chromium**, que é a base do Chrome.

Da mesma forma que o Gecko Driver, deve-se indicar o caminho onde se encontra o executável do drive atribuindo-o à variável **webdriver.chrome.driver**, que se encontra em **serenity.properties** (localizada na pasta raiz de cada projeto).

Também atualizar o drive sempre que possível. Basta então acessar <http://chromedriver.chromium.org/downloads> e baixar o arquivo mais recente de acordo com o sistema operacional.

# Analytics BDD

Sistema desenvolvido para a infra validar os status de algumas aplicações que serão configuradas no arquivo (src/test/resources/stories/aplicacao/valida\_aplicacao.story)

Para fazer o build do projeto siga os passos abaixo:

* Botão direito no projeto "analytics-bdd", Maven > Run As > Maven build...
* No campo "Goals" informe: clean verify
  + OBS.: Caso queira filtrar por categoria/tags, seguem abaixo alguns exemplos;

Executa os testes que contém as tags: **@aplicacao Efetividade**

clean verify -Dmetafilter="+aplicacao Efetividade"

* Aba JRE, marque a opção "Alternate JRE": escolha um jdk, Exemplo: jdk1.8.0\_101

## Procedimentos para execução

Geralmente esses passos são executados uma vez/dia antes de começar as execuções.

* Executar os scripts de reset, base "RELOH":
  + scripts\_sql\_reset\_base\_testes/RESET\_PRODUTOS.sql
  + scripts\_sql\_reset\_base\_testes/RESET\_PLANO\_DE\_PAGAMENTO.sql
* Fechar/Abrir caixa e PDV; Loja: 291, PDV: 110, Base: "RELOH":
  + Pois é utilizado nos testes, caso o dado "receber\_pedido" esteja setado como true em conexao.json;
  + Obs.: Se ao tentar fechar o pdv der o erro: "PDV / ECF possui histórico de uso de NFC-e" devemos executar o sql:

**delete from (select \* from MV\_CONTROLE\_NFCE where NRO\_PDV = 110 and nro\_loja\_pdv = 1101);**

* Fechar/Abrir caixa e PDV; Loja: 4010, PDV: 110, Base: "RELOH"
  + Pois é utilizado no testes de nota fiscal de acompanhamento (NotaFiscalDeAcompanhamento) executar via Jenkins job "NOTA FISCAL DE ACOMPANHAMENTO";
* Ao executar os testes do selenium, observar a instância que está configurada nos arquivos, variável IP;

# Glossário

**Java**: Linguagem de programação. O que será baixado é o JDK(Java Development Kit) que é utilizado para desenvolvimento.

**Eclipse:** Plataforma para desenvolvimento de sistemas.

**Maven:** Ferramenta de automação de build (construção / execução do projeto)

**Git:** Programa de versionamento de sistemas.