

Jenkins

Automatize tudo sem
complicações



© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida

Vivian Matsui

Revisão

Bianca Hubert

Vivian Matsui

[2016]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

AGRADECIMENTOS

Agradeço a você, por investir o seu tempo neste livro para ser mais produtivo!

Agradeço a minha família, a Deus e meus colegas de trabalho por tudo. Muito obrigado.

QUEM É FERNANDO BOAGLIO?

Uma imagem fala mais que mil palavras... Veja quem eu sou na figura:

Já foi estudante...



Linux user desde 1996



Já deu aulas...



Developer+commiter!



E começou a escrever...



BOAGLIO.COM

DEDICATÓRIA

Este livro é dedicado ao saudoso Luiz Arnaldo de Gusmão Bastos, ou simplesmente **Luca Bastos**, um homem que foi e continua sendo um exemplo e inspiração para muitos profissionais de TI.

Por favor, reserve alguns minutos de sua vida e assista a algumas das palestras do Luca, listadas a seguir. Agradeça-me depois. =)

- <http://bit.do/lucabastos1> – Da descoberta do ágil ao manifesto.
- <http://bit.do/lucabastos2> – Evolução e o futuro do desenvolvimento de software.
- <http://bit.do/lucabastos3> – Carreira: como você se imagina em 40 anos?
- <http://bit.do/lucabastos4> – O que é inovação?



PREFÁCIO

Para que fazer na mão se é possível automatizar?

Fazer builds e deploys manuais leva um precioso tempo e, quando tudo fica automatizado com o Jenkins, parece inacreditável o tempo que era gasto fazendo essas tarefas. E você se questiona como não tinha automatizado isso antes.

Público-alvo

Este livro foi feito para desenvolvedores ou operadores de sistemas que desejam otimizar as suas tarefas do dia a dia e automatizar o máximo possível.

Quickstart – não perca tempo

Para rapidamente entender os conceitos mais importantes e disponibilizar o seu ambiente de integração contínua, não será preciso ler todos os capítulos, apenas os cinco primeiros.

Melhorando – explorando as possibilidades

Os capítulos restantes complementam com estratégias de orquestração, conceito de slaves, deploys, customizações e a utilização dos plugins mais importantes.

Hacking – crie o seu plugin do Jenkins

Entenda como funciona internamente o Jenkins e sua API, e aprenda a construir plugins para estender as funcionalidades dessa incrível ferramenta de integração contínua.

Código-fonte

O código-fonte deste livro está disponível em <https://github.com/boaglio/jenkins-casadocodigo>, onde foram criadas tags para cada um dos capítulos.

Como o Jenkins me ajudou

A minha vida era mais tranquila quando eu cuidava de dois sistemas, pois facilmente eu sabia como eles estavam em todos os ambientes. Quando esse número aumentou muito, não demorou para eu me esquecer de fazer alguma tarefa, ou tentar fazer várias ao mesmo tempo e dar tudo errado no final.

Já tive experiência em automatizar algumas coisas com outras soluções, mas a customização era tão penosa que o "fazer na mão" era mais rápido e eficaz. Isso tudo até conhecer o Jenkins. Finalmente, uma ferramenta apareceu para ajudar o meu dia a dia, e não ser mais uma da lista de ferramentas que eu xingava por serem pagas e, mesmo assim, cheias de problemas.

Claro que todo software não é perfeito, afinal, ele é feito pelo ser humano, que comete erros. Entretanto, o que é excelente no Jenkins é a sua comunidade superativa, além de ele ser também de código aberto.

Hoje, com o Jenkins, eu consigo fazer muitas coisas ao mesmo tempo e o que antes era manual e demorava horas, hoje, é apenas uma consulta geral no painel que leva poucos segundos, apenas para conferir se alguma coisa não funcionou. O meu trabalho ficou bem mais produtivo. Espero que o Jenkins ajude sua vida a melhorar também.

Sumário

Jenkins: automatize tudo sem complicações	1
1 Introdução	2
1.1 Integração contínua, eu preciso disso?	3
1.2 Prazer, Jenkins	5
1.3 Arrumando a casa	6
1.4 Instalando	8
1.5 Próximos passos	14
2 Conceitos fundamentais	15
2.1 Job	15
2.2 Build	15
2.3 Pipeline	16
2.4 Plugin	17
2.5 Artifact	17
2.6 Dashboard e view	17
2.7 Executor	18
2.8 Nó master e slave	18
2.9 Workspace	18
2.10 Visão geral da tela inicial	19
2.11 Próximos passos	20

3 Builds	21
3.1 Pacote web	21
3.2 Configurando o básico	23
3.3 Nosso primeiro build com Maven	28
3.4 Nosso primeiro build com ANT	31
3.5 Próximos passos	35
4 Publicando pacotes no Artifactory	36
4.1 Em que o Artifactory ajuda?	36
4.2 Instalando plugin do Artifactory	38
4.3 Configurando o plugin do Artifactory	39
4.4 Publicando pacotes no Artifactory	40
4.5 Verificando o pacote publicado	42
4.6 Próximos passos	43
5 Criando uma pipeline de entregas	45
5.1 Publicando em aceite	45
5.2 Publicando em produção	49
5.3 Ligando os pontos	49
5.4 Agendando builds	53
6 Autenticação e segurança	56
6.1 Tipos de autenticação	57
6.2 Tipos de autorização	58
6.3 Recuperando o acesso perdido	59
6.4 Próximos passos	60
7 Validando e atualizando o banco de dados	61
7.1 Instalando os plugins	61
7.2 Usando o script de validar o horário	62
7.3 Flyway	66

7.4 Atualizando a base de dados de aceite	67
7.5 Atualizando a base de dados de produção	68
7.6 Atualizando a pipeline	69
7.7 Próximos passos	70
8 Aumentando a qualidade das entregas	72
8.1 Testando o software antes	72
8.2 Gerando as métricas no Sonar	73
8.3 Atualizando a pipeline	74
8.4 Próximos passos	75
9 Promovendo suas entregas	77
9.1 Instalando o plugin	77
9.2 Promovendo um job	78
9.3 Acessando o job sem permissão	80
9.4 Acessando o job com permissão	81
9.5 Próximos passos	82
10 Testando sua aplicação	84
10.1 Instalando o plugin	84
10.2 Ativando o job para usar o Selenium	85
10.3 Selenium	86
10.4 Testando fora do servidor com o Jenkins slave	87
10.5 Configurando testes no Internet Explorer	92
10.6 Próximos passos	94
11 Plugins ninja e dicas	95
11.1 Plugins usados neste livro	95
11.2 Plugins recomendados	97
11.3 Próximos passos	99
12 Criando o seu plugin	100

12.1 Jelly	100
12.2 Criando um plugin simples	100
12.3 Codificando o plugin	101
12.4 Instalando o plugin	106
12.5 Executando o plugin	106
12.6 Próximos passos	107
13 # Continue seus estudos	108
 Apêndices	 110
14 Apêndice A – Instalando o SVN	111
15 Apêndice B – Instalando o Artifactory	115
16 Apêndice C – Maven	119
16.1 O que é o Maven e como usá-lo?	119
16.2 Como instalá-lo?	121
16.3 Fases do Maven	121
17 Apêndice D – Sonar	123
17.1 O que é o Sonar e como usá-lo?	123
17.2 Como instalar?	124
17.3 Iniciando o Sonar	125
17.4 Melhorando o Sonar	126
18 Apêndice E – Selenium	127
18.1 O que é o Selenium e como usá-lo?	127
19 Apêndice F – Gradle e Groovy	129
19.1 O que é o Groovy?	129
19.2 O que é o Gradle?	130
20 Apêndice G – Flyway	133

20.1 O que é o Flyway e como usá-lo?	133
20.2 Como funciona?	133
21 Apêndice H – Cenário geral	136
21.1 Máquinas do ambiente	136
21.2 Servidores do sistema	136
21.3 Outros servidores	137

Jenkins: automatize tudo sem complicações

INTRODUÇÃO

Não existe inovação sem implementação. Luca Bastos

Bruno acabou de casar e gosta do que faz. Ele trabalha com TI em uma pequena empresa de câmbio e não tem muitos sistemas para cuidar, mas os poucos que tem são essenciais para o negócio da empresa.

No dia a dia, Bruno tem uma vida corrida e tem suas dúvidas se é tão produtivo assim. A sua empresa tem uma aplicação de controle de câmbio feita em Java, chamada ::Minhas Moedas::.

Para fazer a alteração mais simples possível, por exemplo, mudar um texto na tela de "Bom dia" para "Olá", Bruno tem muito trabalho para fazer essa mudança chegar lá no ambiente de produção. Ele precisa:

1. Alterar o seu código e commitar a mudança no seu sistema de controle de versão (SVN, Git etc.);
2. Gerar um pacote de sua aplicação web;
3. Gerar um arquivo .zip de backup dos fontes;
4. Atualizar o banco de dados de homologação;
5. Fazer o deploy do pacote no ambiente de homologação;
6. Fazer os testes manualmente para validar a alteração em homologação;
7. Atualizar o banco de dados de produção;
8. Fazer o deploy do pacote no ambiente de produção;

9. Fazer os testes manualmente para validar a alteração em produção;
10. Enviar um e-mail para o seu chefe avisando que está tudo pronto.

Quando Bruno avalia suas realizações, percebe que ficou mais tempo fazendo operações manuais do que implementando melhorias no sistema ::Minhas Moedas::. Isso soa familiar?

Provavelmente sim, muitas tarefas como as do Bruno consomem o tempo do nosso dia a dia. Uma solução para esse problema é automatizar a maior parte delas, assim podemos focar no que realmente interessa.

No mercado, existem algumas soluções consolidadas. Vamos falar aqui de uma das mais usadas: Jenkins CI (*Jenkins Continuous Integration*), ou simplesmente Jenkins.

1.1 INTEGRAÇÃO CONTÍNUA, EU PRECISO DISSO?

O termo IC (Integração Contínua) passa a ideia de algo junto (integrado) e feito sem parar (continuamente). Mas o que exatamente tem esse comportamento?

A resposta é: o ciclo de desenvolvimento de um sistema. Começamos criando código, depois commitamos (guardamos) tudo em um repositório (em sua maioria SVN ou Git), testamos a aplicação, depois publicamos o sistema gerado desse código em algum lugar e, finalmente, passamos para o usuário usar. Depois, temos alterações no sistema, que podem ser melhorias ou correções de bugs, onde voltamos a criar código. E assim estamos de volta à primeira etapa do ciclo de desenvolvimento.

Podemos resumir esse processo de desenvolvimento na figura:

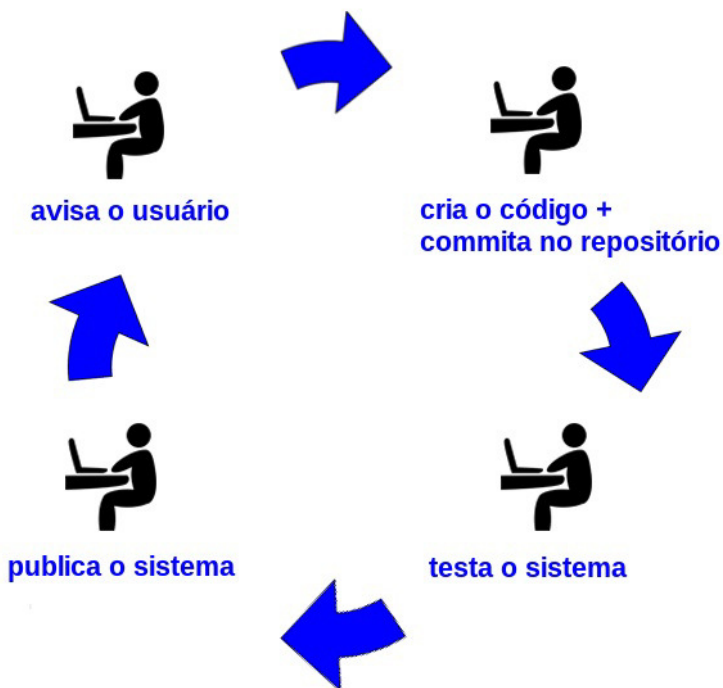


Figura 1.1: Ciclo de desenvolvimento sem o Jenkins

Compare com a próxima figura, do mesmo ciclo com o Jenkins:



Figura 1.2: Ciclo de desenvolvimento com o Jenkins

Note que na primeira figura, o desenvolvedor era responsável por todas as etapas do ciclo, já na segunda, ele fazia apenas uma delas, pois o Jenkins fazia todo o resto!

Esse é o princípio da IC com o Jenkins: o desenvolvedor faz a tarefa criativa (o código), e o Jenkins faz as tarefas repetitivas (testes, deploy etc.). Outra vantagem é que, com quase tudo automatizado, podemos ter um sistema atualizado diversas vezes no mesmo dia, e não poucas vezes ao mês, como na maioria dos sistemas.

1.2 PRAZER, JENKINS

O Jenkins é um servidor de integração contínua open source e é feito em Java, que disponibiliza mais de 1.000 plugins para suportar

construção (*build*) e testes de virtualmente qualquer tipo de projeto.

Com ele, é possível automatizar uma série de procedimentos, como deploy em um servidor, rodar um conjunto de testes, rodar scripts em um banco de dados, atualizar uma aplicação web, e até delegar a execução para outros servidores (o que é chamado de slave).

Ele existe há mais de 10 anos no mercado e, em 2008, recebeu o prêmio *Duke's Choice Award* na categoria Developer Solutions. Em 2011, o seu criador Kohsuke Kawaguchi recebeu o prêmio *Google-O'Reilly Open Source Award*.

1.3 ARRUMANDO A CASA

Vamos melhorar a vida de Bruno e aprender muita coisa com isso. Antes de começar, vamos juntar tudo o que precisamos.

Não tem nada mais chato que no meio de uma reforma do que você precisar sair para comprar alguma coisa que está faltando, ou no meio de uma receita você perceber que esqueceu de comprar um ingrediente. Então, instalaremos tudo o que precisamos, e depois aprenderemos como usar da melhor maneira possível.

O básico

Sendo o Jenkins feito em Java, naturalmente ele precisa de uma JVM para funcionar. Recentemente, o projeto Jenkins decidiu encerrar o suporte ao Java 1.6, portanto, qualquer versão 1.7 ou superior será suficiente. Entretanto, a recomendação oficial é de sempre usar a última versão estável JDK disponível.

Como todo servidor, é sugerido que use o sistema operacional Linux, mas existe a opção de instalar em Mac OS X ou Microsoft Windows, que inclusive já instala a JVM junto e cria um serviço

também.

As ferramentas

Para o Jenkins usar todo o seu potencial, ele precisa de algumas ferramentas para fazer o build (construir) suas aplicações, como o ANT, o Maven ou diferentes versões da JDK. Podemos usar suas versões já instaladas, ou solicitar que a ferramenta instale automaticamente.

Além disso, será necessário acessar um servidor de controle de versão para buscar o código-fonte das aplicações (CVS, SVN, Git). Para um exemplo de instalação, consulte o Apêndice A.

Se desejarmos armazenar os pacotes gerados em um gerenciador de repositórios, será necessário também ter instalado algum deles: Apache Archiva, Sonatype Nexus ou Artifactory (para um exemplo de instalação, consulte o Apêndice B.) Outras opções estão disponíveis, como ter um servidor Active Directory para delegar a autenticação do Jenkins, ou um servidor SMTP para envio de e-mails.

Qual versão instalar?

Felizmente, a instalação é um processo muito simples, pois o projeto Jenkins proporciona instaladores nativos para várias plataformas, como Linux, Windows e Mac OS X. Entretanto, por ser feito em Java, ele pode rodar, por exemplo, em uma máquina Solaris sem problemas.

Todos os arquivos para download podem ser baixados do site <http://jenkins-ci.org/>, porém existem duas versões diferentes para baixar:

- *Release* – é a versão estável mais atual, sai uma versão a

cada semana;

- *LTS Release* ou *Long Term Support Release* – é a versão estável mais conservadora, sai uma a cada doze semanas (é essa a versão que será utilizada neste livro).

Em ambiente de produção, é sugerido o uso da LTS, por ser uma versão mais testada pela comunidade. Além disso, quando um bug aparece em um plug-in, o responsável às vezes testa apenas com a última LTS. E caso funcione, ele não arruma o bug e apenas confere se na próxima LTS o problema se repete.

1.4 INSTALANDO

Antes de começarmos a falar sobre instalação, é importante entender que se trata da parte servidora (*server side*), pois o Jenkins permite delegar parte de seus trabalhos para estações de outros sistemas operacionais. Por exemplo, podemos ter um servidor Jenkins rodando tarefas nele e, ao mesmo tempo, enviando tarefas para uma estação (slave) Windows, ou MacOSX, ou Linux.

A instalação em Linux Ubuntu

É sugerido na instalação em Linux usar o mesmo sistema de pacote da distribuição, assim fica fácil de trabalhar com o Jenkins se ele for instalado como serviço e também deixamos automática a verificação de versão mais nova.

Para cadastrarmos a chave e o repositório do APT do Jenkins no Ubuntu, rodamos os comandos:

```
> wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key  
| sudo apt-key add -  
OK  
> sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/  
> /etc/apt/sources.list.d/jenkins.list'  
>
```

Em seguida, executamos os comandos para instalar:

```
> sudo apt-get update  
> sudo apt-get install jenkins
```

O Jenkins sobe como padrão na porta 8080. Para alterarmos esse valor, vamos editar o arquivo `/etc/default/jenkins` e reiniciamos o serviço com:

```
> sudo service jenkins restart
```

A tela inicial do Jenkins está disponível em <http://localhost:8080/>, com um resultado semelhante à figura:

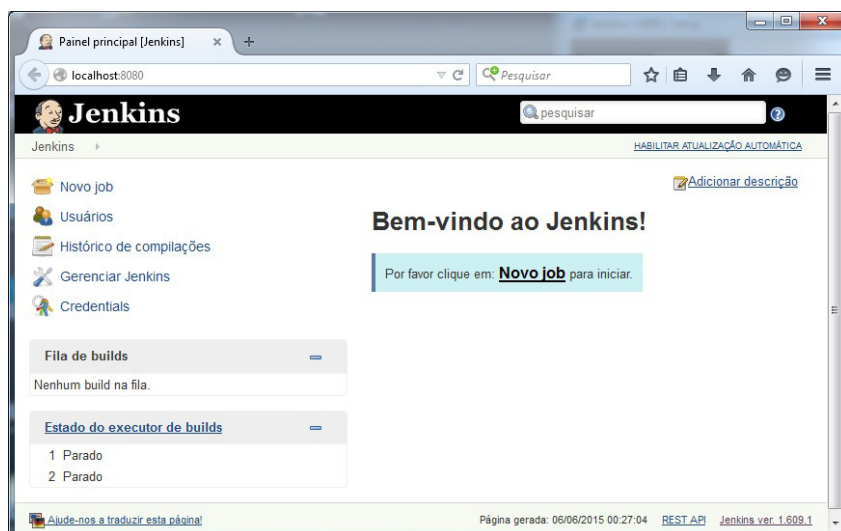


Figura 1.3: Tela inicial do Jenkins

A documentação oficial está em <http://bit.do/jenkinsNoLinux>.

A instalação em Mac OS X

A instalação é feita com o arquivo do tipo `pkg` disponível para download.

Se for necessário alterar alguma configuração do serviço, edite o

arquivo `/Library/Preferences/org.jenkins-ci` e execute os comandos:

```
> sudo launchctl load  
  /Library/LaunchDaemons/org.jenkins-ci.plist  
> sudo launchctl unload  
  /Library/LaunchDaemons/org.jenkins-ci.plist
```

Depois acessamos a tela inicial do Jenkins em <http://localhost:8080/>, com um resultado semelhante à última figura.

A documentação oficial está em <http://bit.do/jenkinsNoMacOSX>.

A instalação em Microsoft Windows

Depois de baixar o arquivo do site, iniciamos a instalação:

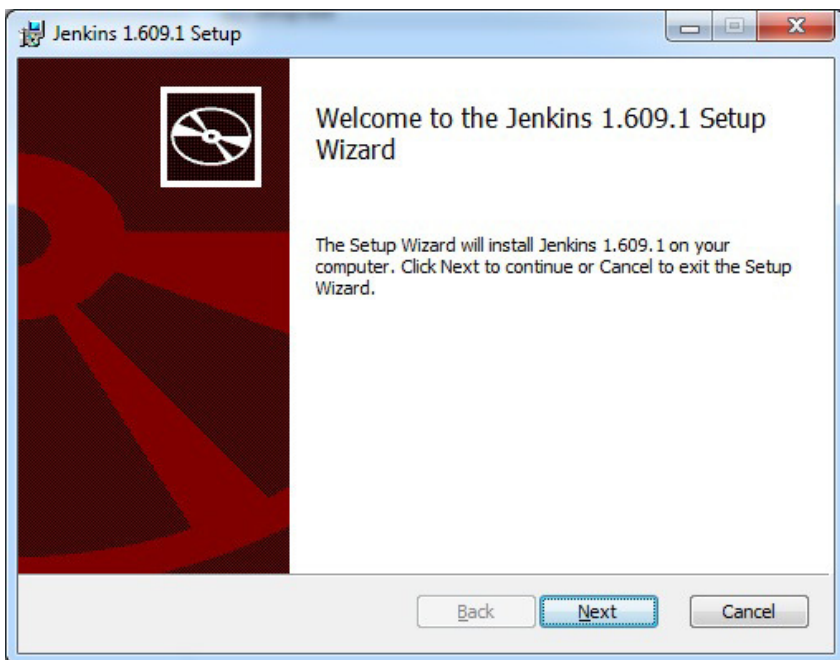


Figura 1.4: Tela inicial da instalação

Em seguida alteramos o diretório padrão para a raiz, como por exemplo, C:\Jenkins\ .

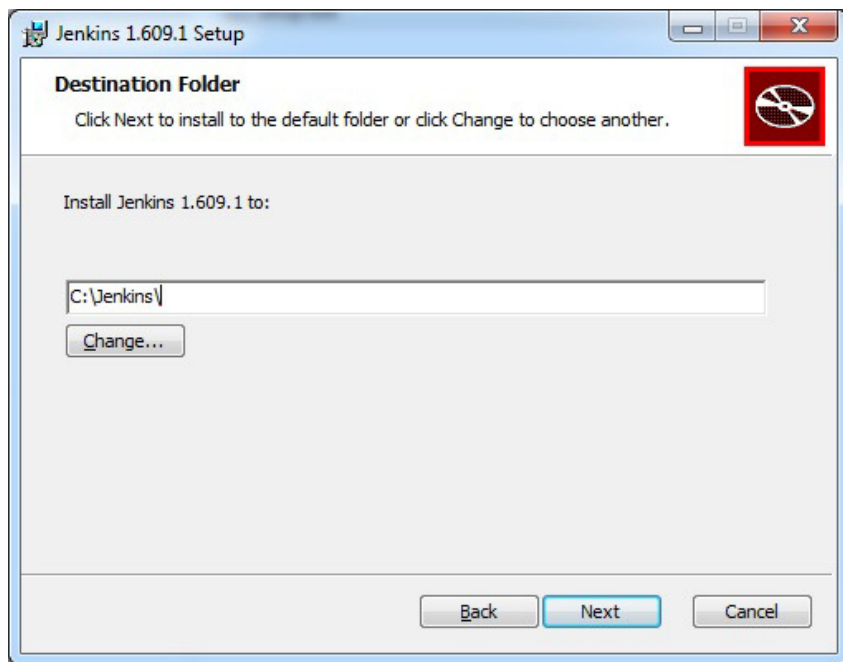


Figura 1.5: Escolhendo o diretório do Jenkins

E começamos a cópia dos arquivos para o diretório escolhido:

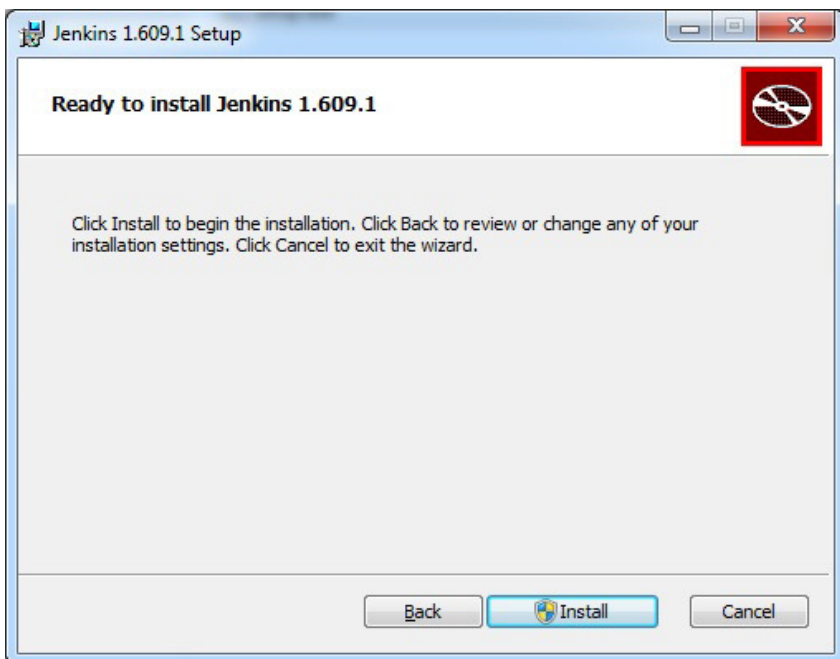


Figura 1.6: Iniciando a instalação

Clique em `finish` para concluir a instalação.

Assim, acessamos a tela inicial do Jenkins em <http://localhost:8080/>. A instalação cria um serviço automático chamado Jenkins e já colocar no ar:

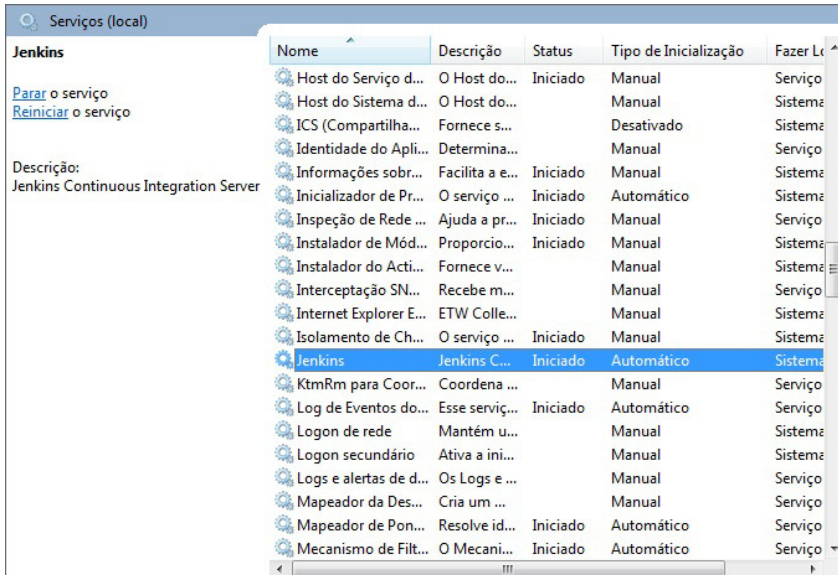


Figura 1.7: Serviço do Jenkins instalado

Acessamos a tela inicial do Jenkins em <http://localhost:8080/>, com um resultado semelhante aos anteriores.

A documentação oficial está em <http://bit.do/jenkinsNoWindows>.

Instalação via Docker

Para os usuários Docker, existe a imagem oficial do Jenkins disponível para uso. A sua instalação pode ser feita por meio do comando a seguir:

```
docker run -p 8080:8080 jenkins
```

A imagem oficial será baixada e mapeada para a porta 8080.

Para mais informações sobre a configuração e uso, acesse o endereço https://registry.hub.docker.com/_/jenkins/.

Banco de dados

Chegamos ao final do capítulo, mas onde está a configuração do banco de dados?

Bom, o Jenkins não utiliza nenhum banco de dados, ele grava todas as suas informações em disco, no diretório conhecido como `JENKINS_HOME` – que pode ser `<DIRETÓRIO-HOME>/jenkins` ou `/var/lib/jenkins` em algumas distribuições Linux. Portanto, pensando em backup, esse será o diretório que devemos ter cópia.

1.5 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- Quais as principais vantagens do Jenkins;
- Como instalar o Jenkins.

Bruno agora entende que precisa investir o seu tempo em automatizar suas tarefas, e com certeza esse tempo será recompensado quando ele deixar de fazer as tarefas chatas que faz frequentemente.

No próximo capítulo, vamos aprender rapidamente os principais conceitos do Jenkins, algo necessário para tirar o máximo proveito dele.

CONCEITOS FUNDAMENTAIS

Não fazer somente software bem feito, mas feito a partir de claro entendimento. Luca Bastos

Bruno tem muita coisa para automatizar (como deploys e atualizações de banco), mas inicialmente ele precisa entender os principais conceitos do Jenkins para conseguir encaixar as suas necessidades em tudo o que essa ferramenta pode lhe oferecer.

Vamos usar o Jenkins para automatizar algumas tarefas, ou seja, fazer o trabalho sujo para nós. Alguns conceitos são fundamentais para o melhor entendimento da ferramenta.

2.1 JOB

Um **job** é uma tarefa que o Jenkins deve fazer. Normalmente, tem alguns parâmetros de execução, juntamente com alguns procedimentos que podem ser feitos antes ou depois de sua execução.

Por exemplo, um job de build de uma aplicação web, ou job de execução de um conjunto de testes. O mesmo job normalmente tem ou um mais builds.

2.2 BUILD

É a construção, uma execução de um job (tarefa), por exemplo o procedimento de montar um pacote, que pode envolver download, compilação ou testes.

O build pode ser considerado como uma instância de um job, como o último processo de um job de deploy. Ele também possui diferentes status:

	Falha no build
	Build instável
	Build feito com sucesso
	Build pendente (esperando)
	Build cancelado
	Build desligado

Figura 2.1: Tipos de status de build

2.3 PIPELINE

Um job pode ter dependência com outro job; se um for acionado, ao terminar, pode chamar o outro automaticamente, criando uma cadeia de jobs enfileirados, parecendo com uma "linha de canos" (*pipeline*, figura a seguir). Um exemplo é um job de build da aplicação web estar ligado ao job de deploy dessa mesma aplicação.



Figura 2.2: Pipeline

2.4 PLUGIN

A integração do Jenkins, o nosso superexecutor de tarefas, com os diversos tipos de servidores e sistemas é feita por meio de plugins, não só para melhorar um funcionamento existente, mas também para criar um que não existe.

2.5 ARTIFACT

Artifact (ou artefato) é o pacote resultante de um build executado com sucesso. Ele pode ser um arquivo `pom.xml` , `JAR` , `WAR` , `EAR` etc.

2.6 DASHBOARD E VIEW

O Dashboard (ou painel principal) é o local em que temos uma visão completa da execução de todos os jobs do Jenkins. Ele pode ser dividido em views específicas de alguns jobs, mas tem como objetivo dar uma visão como um todo.

Ele possui ícones para cada job relacionados ao clima: se temos um Sol, está tudo bem, e se temos uma tempestade, temos problemas (figura a seguir).

	Nenhum build recente falhou
	20 a 40% dos builds recentes falharam
	40 a 60% dos builds recentes falharam
	60 a 80% dos builds recentes falharam
	Todos os builds recentes falharam

Figura 2.3: Tipos do status do job

2.7 EXECUTOR

São *threads* que rodam os builds dos jobs. O Jenkins permite que ajuste esse número de executores conforme a sua necessidade, sendo que, por padrão, são configurados apenas dois. Normalmente, um número interessante é um executor para cada core da máquina, por exemplo, uma máquina quadcore trabalharia bem com quatro executores.

2.8 NÓ MASTER E SLAVE

O nó *master* (mestre) roda o Jenkins e controla nós *slaves* (escravos), que são as máquinas auxiliares conectadas.

2.9 WORKSPACE

É a área de trabalho existente em cada job, onde o Jenkins baixa os arquivos necessários e roda os processos solicitados.

2.10 VISÃO GERAL DA TELA INICIAL

A tela inicial do Jenkins pode ser dividida em cinco partes, como ilustrado na figura a seguir. Note que essa tela pode ser customizada depois com alguns plugins, mas vamos comentar a opção padrão.



Figura 2.4: Visão geral da tela inicial do Jenkins

A – Gerenciamento

Essa é a parte principal do Jenkins, com as opções de:

- **Novo job** – criar uma nova tarefa de build, deploy ou outra coisa;
- **Usuários** – gerenciamento de usuários (após a instalação, vem desativado);
- **Histórico de compilações** – exibe um gráfico de histórico dos builds;
- **Gerenciar Jenkins** – exibe as opções do painel de controle do Jenkins;
- **Credentials** – gerenciamento de certificados digitais.

B – Fila de builds

Aqui são exibidos os builds na fila de espera, esperando para

executar.

C – Estado do executor de builds

Aqui são exibidos os builds em execução, com algumas opções de visualizar o build em detalhes ou cancelar.

D – Habilitar atualização automática

Como o Jenkins constantemente está rodando builds, é uma interessante opção ativar a atualização automática e deixar o browser atualizar automaticamente a página, exibindo a situação atual de todos os builds.

E – Adicionar descrição

É a opção de adicionar um texto descritivo na página, podendo usar algumas tags de HTML.

2.11 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- Os termos e conceitos principais do Jenkins;
- As opções da página inicial.

Antes de sair configurando as coisas sem saber direito o que está fazendo, Bruno resolveu parar um pouco e entender os conceitos do Jenkins. Isso será muito útil para usar todo o potencial do Jenkins e, assim, auxiliar em seu trabalho.

BUILDS

É preciso reservar tempo e esforço em prol da melhoria contínua.

Luca Bastos

Vamos criar algumas tarefas para automatizar algumas coisas que Bruno faz manualmente, como:

- Gerar um pacote de sua aplicação web;
- Gerar um arquivo `.zip` de backup dos códigos-fontes.

3.1 PACOTE WEB

Vamos iniciar clicando na opção `Novo job` da tela inicial. Preenchemos com o nome do job `minhasmoedas-war` e escolhemos a opção de `Construir um projeto maven`.

Vamos adicionar o local de origem dos códigos-fontes, ou seja, a URL do SVN de onde os arquivos serão baixados:

Gerenciamento de código fonte

- ☐ Nenhum
 - ☐ CVS
 - ☐ CVS Projectset
 - ☒ Subversion
- Módulos

Repository URL

http://cascao/svn/boagliorep/minhasmoedas/ ?

Credentials

- none - Add

Local module directory

.

Repository depth

infinity

Ignore externals

☒

Figura 3.1: Configuração do caminho dos fontes

Diversas opções vêm preenchidas, e se algo estiver errado, o Jenkins avisa em vermelho. Em uma primeira criação de job baseado no Maven, o Jenkins alerta que não encontrou onde está o Maven e que será preciso configurar.

Trigger de builds

☒ Construir um SNAPSHOT sempre que executar uma construção ?
☐ Construir após a construção de outros projetos ?
☐ Construir periodicamente ?
☐ Consultar periodicamente o SCM ?

Pre Steps

Add pre-build step

Construir

Versão do Maven

Jenkins precisa saber onde está a instalação do Maven
Favor fazer [a configuração do sistema.](#)

POM Raiz

pom.xml ?

Mathe a nomee

Figura 3.2: Aviso da configuração incompleta do Maven

Mesmo incompleta, podemos gravar o job clicando em salvar .

Outro detalhe interessante é o conceito da opção Credentials (credenciais). Existe uma grande chance de que uma combinação de usuário e senha seja reutilizada em alguma configuração. Pensando

nisso, foi criado um conceito onde as senhas são cadastradas e depois, onde for necessário, é feita a referência.

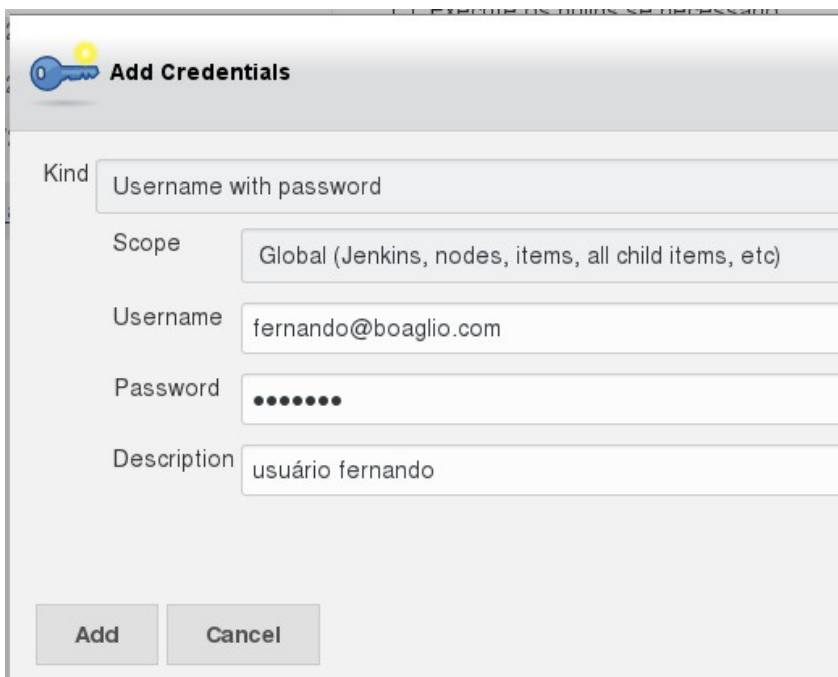


Figura 3.3: Cadastrando uma nova credencial

Gerenciamento de código fonte

- ☐ Nenhum
 - ☐ CVS
 - ☐ CVS Projectset
 - ☒ Subversion
- Módulos

Repository URL	http://cascao/svn/boagliorep/minhasmoedas
Credentials	fernando@boaglio.com/***** (usuário fernando)

Figura 3.4: Referenciando a credencial cadastrada no job

3.2 CONFIGURANDO O BÁSICO

Vamos fazer alguns ajustes mínimos para o Jenkins estar apto para executar os nossos builds.

Clicando na opção `configurar` , detalhamos novamente as configurações do job que acabamos de gravar:

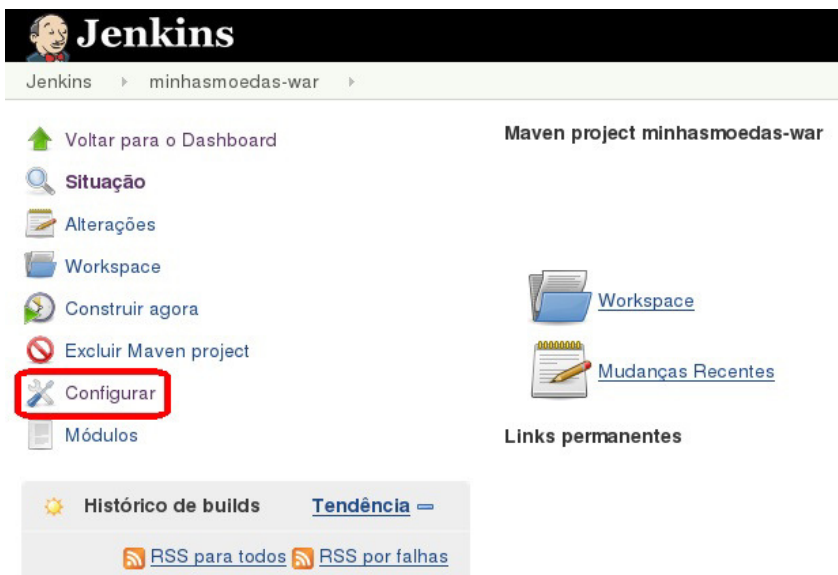


Figura 3.5: Configurar job no Jenkins

Agora, vamos clicar na opção destacada `Configuração` do sistema para ajustar o que o Jenkins está reclamando. Essa mesma opção pode ser acessada pela tela principal na opção `Gerenciar Jenkins` e, em seguida, `Configurar o sistema` .

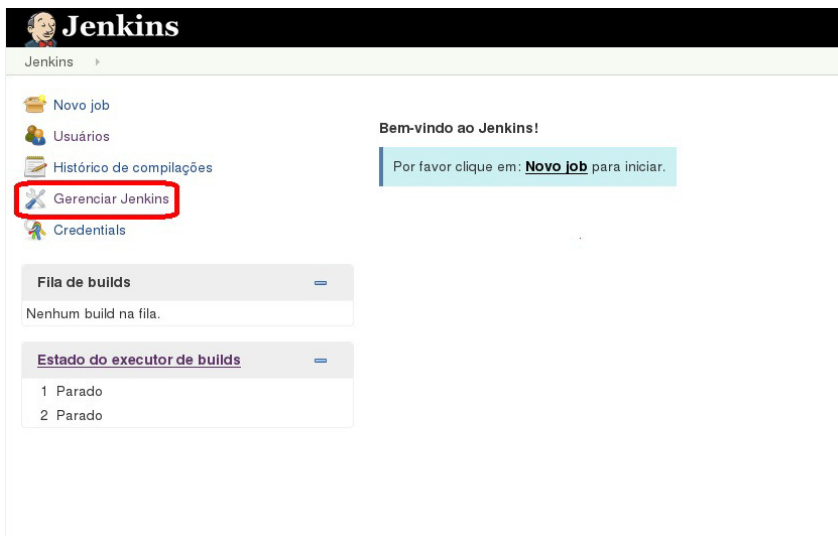


Figura 3.6: Configuração geral do Jenkins – passo 1

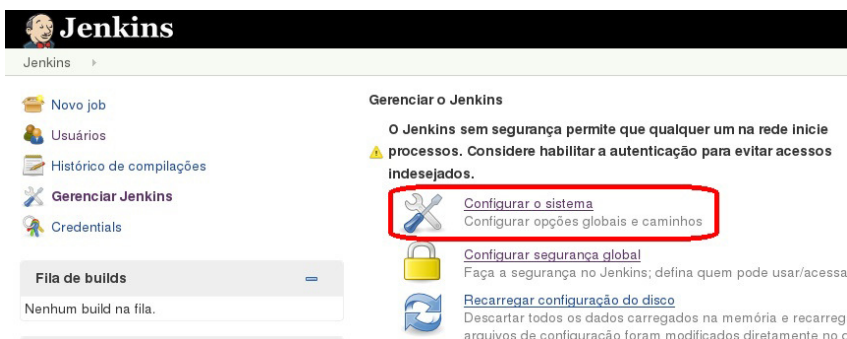


Figura 3.7: Configuração geral do Jenkins – passo 2

AS DIFERENTES CONFIGURAÇÕES É importante entender a diferença das duas configurações: a configuração geral vale para todos os jobs existentes do Jenkins, e deve ter um cuidado muito especial; enquanto as configurações de cada job afetam apenas o job alterado.

Na figura a seguir, vemos as opções de instalação dos componentes básicos do Maven.

Propriedades globais

☐ Ferramentas locais

☐ Variáveis de ambiente

Maven Configuration

Default settings provider: Usar as configurações padrão do Maven ▼

Default global settings provider: Usar as configurações globais padrão do Maven ▼

JDK

JDK instalações: Adicionar JDK

Lista de JDK instalações nesse sistema

Ant

Ant instalações: Adicionar Ant

Lista de Ant instalações nesse sistema

Maven

Maven instalações: Adicionar Maven

Lista de Maven instalações nesse sistema

Configuração do Projeto Maven

Global MAVEN_OPTS: [] ▼ ?

Local Maven Repository: Default (~/.m2/repository) ▼ ?

Figura 3.8: Instalar componentes básicos

A JDK, ANT e Maven podem ser usados nas versões instaladas no sistema operacional, entretanto, uma atualização nessas versões pode ter resultados indesejáveis no Jenkins. Dessa forma, é interessante que exista um controle de qual versão será usada.

A instalação pode ser feita automaticamente pelo Jenkins, ou referenciando um diretório local. Na figura a seguir, configuramos a instalação automática do JDK do Java 8:

JDK
JDK instalações

JDK

Nome

1.8

☒ Instalar automaticamente

Instalar de java.sun.com

Versão

Java SE Development Kit 8u45

☒ Eu concordo com a licença do Java SE Development Kit

Excluir instalador

Adicionar instalador

Adicionar JDK

Lista de JDK instalações nesse sistema

Figura 3.9: Instalar JDK automaticamente

Já na figura seguinte, instalamos o ANT apontando para um diretório existente onde ele já está instalado:

Ant
Ant instalações

Ant

Nome

1.9.5

ANT_HOME

/opt/apache-ant-1.9.5

☐ Instalar automaticamente

Excluir Ant

Adicionar Ant

Lista de Ant instalações nesse sistema

Figura 3.10: Instalar ANT apontando para um diretório local

Vamos deixar o Maven para instalar automaticamente também:

Maven
Maven instalações

Maven

Nome

3.3.3

☒ Instalar automaticamente

Instalar a partir do Apache

Versão

3.3.3

Excluir instalador

Adicionar instalador

Adicionar Maven

Lista de Maven instalações nesse sistema

Figura 3.11: Instalar Maven automaticamente

Depois de clicar em `Save` , ficamos com a impressão de que o Jenkins fosse baixar os arquivos, mas na verdade ele somente os baixará quando for realmente necessário, ou seja, quando alguém solicitar um build.

3.3 NOSSO PRIMEIRO BUILD COM MAVEN

Na tela inicial, vamos solicitar a criação de um build, clicando no ícone da extrema direita na linha do job `minhasmoedas-war` :

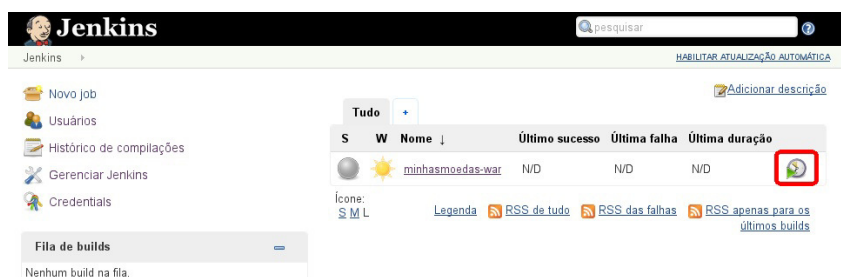


Figura 3.12: Executar o build

Alguns segundos após clicar nessa opção, observamos que à esquerda aparecerá o nome job ao lado de um número do build com `#` , que é um valor crescente e sequencial.

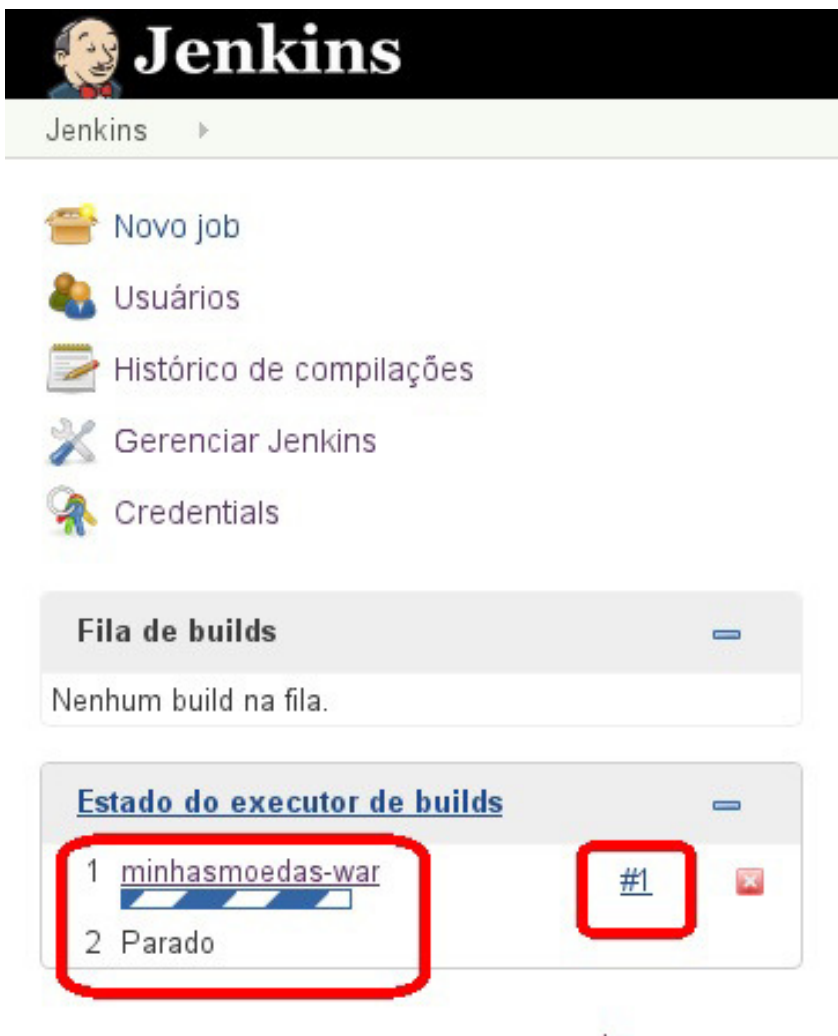


Figura 3.13: Build rodando

Clicando no link do build, temos alguns detalhes, como o tempo de execução do build:



Figura 3.14: Detalhe do build rodando

Conseguimos ver o Jenkins trabalhando clicando em Saída do console :

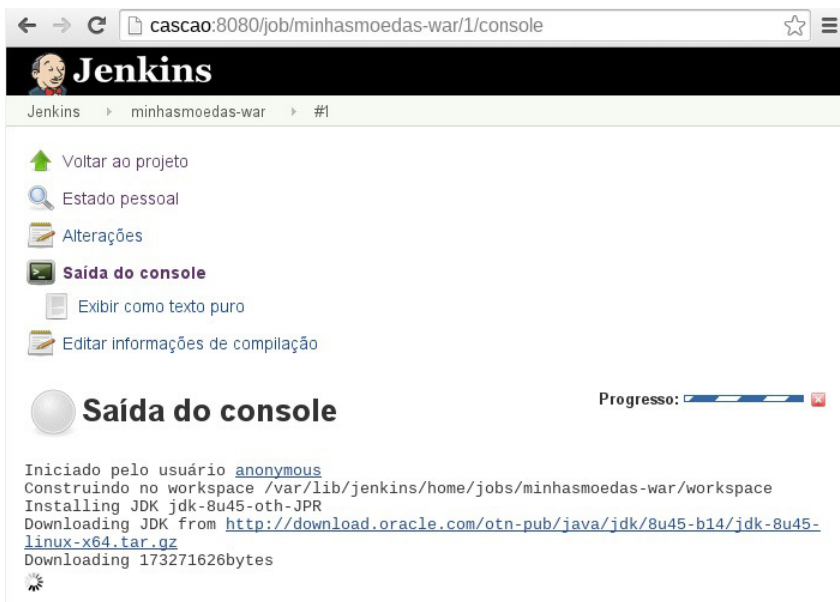


Figura 3.15: Output do build rodando

Será exibido o texto da saída do build e atualizado

automaticamente. Se tudo ocorreu bem, o final da saída será exibido algo assim:

```
[JENKINS] Archiving minhas-moedas.war  
channel stopped  
Finished: SUCCESS
```

Bruno já conseguiu automatizar a sua primeira tarefa com esse job, e já consegue gerar um pacote de sua aplicação web com um clique do seu mouse. Isso é só o começo, vamos ver o que mais podemos automatizar.

3.4 NOSSO PRIMEIRO BUILD COM ANT

Temos no projeto *Minhas Moedas* um processo de backup dos fontes, que é apenas um zip dos arquivos copiados para um diretório específico.

Dentro dos fontes do projeto, temos o arquivo `build.xml` bem simples que possui apenas uma tarefa.

```
<?xml version="1.0" encoding="UTF-8"?>  
<project name="minhasmoedas-fontes-bkp"  
    default="bkp"  
    basedir=".">  
  
    <property name="projeto" value="${ant.project.name}" />  
    <property name="bkp-dir" value="/servidor-bkp/fontes/" />  
    <property name="diretorio-para-zipar" value="src" />  
  
    <target name="clean">  
        <delete file="${projeto}.zip" />  
    </target>  
  
    <target name="bkp">  
        <zip destfile="${projeto}.zip">  
            <fileset dir="${diretorio-para-zipar}" />  
        </zip>  
        <copy file="${projeto}.zip" todir="${bkp-dir}" />  
    </target>  
  
</project>
```

Vamos criar um novo job chamado `minhasmoedas-bkp-fontes`, do tipo `free-style`.

Nome do job

☒ **Construir um projeto de software free-style**
Esta é a central de funcionalidades do Jenkins. Ele construirá seu projeto, e qualquer sistema de builds, e ele até mesmo pode ser usado para outras jobs

☐ **Construir um projeto maven**
Construir um projeto maven. Jenkins tira vantagem de seus arquivos POM. Ainda é um trabalho em progresso, mas disposto a aceitar feedback.

☐ **Construir projeto de múltiplas configurações**
Apropriado para projetos que necessitam de grande número de diferentes ambientes, builds para plataformas específicas, etc.

☐ **External Job**
Este tipo de trabalho permite que você grave a execução de um processo numa máquina remota.) Isto foi projetado para que você possa usar Jenkins sistemas de automação. Veja a documentação [para mais detalhes](#).

☐ **Copiar job existente**
Copiar de

Figura 3.16: Novo job para usar ANT

Em seguida, precisamos informar de onde vêm os arquivos fontes. Para isso, colocamos as mesmas informações do SVN que o job criado anteriormente. Depois informamos que, no processo de build, vamos invocar o ANT, conforme a figura:

Trigger de builds

- ☐ Construir após a construção de outros projetos
- ☐ Construir periodicamente
- ☐ Consultar periodicamente o SCM

Build

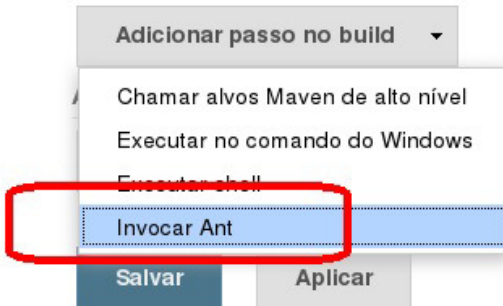


Figura 3.17: Adicionando a chamada ao ANT dentro do job

Escolhemos a versão do ANT (ou deixamos a padrão mesmo), e opcionalmente informamos o nome do `target` (alvo) do ANT. Se existir apenas uma versão instalada, tanto faz escolher a versão padrão ou não, ele usará a mesma.

No nosso arquivo, ele chama-se `bkp`, mas como ele é o `target` padrão, não precisamos colocar o seu nome.

Build

Invocar Ant

Versão do Ant

1.9.5

Alvos

Salvar

Aplicar

Figura 3.18: Escolhendo a versão do ANT e o target dentro do job

Clicamos em **Salvar** para criar o job e, em seguida, podemos iniciar o primeiro build, de maneira semelhante ao job criado anteriormente.

Depois, verificando a saída do console desse build (também semelhante ao job anterior), observamos que o arquivo `zip` foi gerado com sucesso:

```
A      src/main/java
A      src/main/java/com
A      src/main/java/com/boaglio
A      src/main/java/com/boaglio/minhasmoedas
A      src/main/java/com/boaglio/minhasmoedas/service
A      src/main/java/com/boaglio/minhasmoedas/service/CambioService.java
A      src/main/java/com/boaglio/minhasmoedas/type
AU     src/main/java/com/boaglio/minhasmoedas/type/Paginas.java
A      src/main/java/com/boaglio/minhasmoedas/controller
AU     src/main/java/com/boaglio/minhasmoedas/controller/MainController.java
At revision 20
no change for http://cascao/svn/boagliorep/minhasmoedas since the previous build
[workspace] $ /opt/apache-ant-1.9.5/bin/ant
Buildfile: /var/lib/jenkins/home/jobs/minhasmoedas-bkp-fontes/workspace/build.xml

bkp:
[zip] Building zip: /var/lib/jenkins/home/jobs/minhasmoedas-bkp-fontes/worksp
fontes-bkp.zip
[copy] Copying 1 file to /tmp/bkp

BUILD SUCCESSFUL
Total time: 0 seconds
Finished: SUCCESS
```

Figura 3.19: Resultado do job com ANT

Finalmente, temos dois jobs rodando no Jenkins, um com o Maven e outro com ANT.

3.5 PRÓXIMOS PASSOS

Os próximos jobs serão variações desses criados neste capítulo com algumas particularidades, mas o esquema macro é o mesmo.

Certifique-se de que aprendeu como:

- Configurar JDK;
- Configurar ANT;
- Configurar Maven;
- Criar um job com Maven;
- Criar um job com ANT;
- Executar um build;
- Acompanhar a execução de um build.

Agora, Bruno tem dois processos automatizados: um que gera um pacote de sua aplicação web e outro que faz um backup de seus arquivos. Entretanto, ambos precisam ser acionados manualmente no Jenkins para iniciar, então vamos aprender em breve como agendar esses jobs. No próximo capítulo, vamos aprender a automatizar o armazenamento das versões dos sistemas.

PUBLICANDO PACOTES NO ARTIFACTORY

Não somente adicionar valor continuamente, mas só fazer o que adiciona valor. Luca Bastos

4.1 EM QUE O ARTIFACTORY AJUDA?

Em toda empresa, certamente existe pelo menos um tipo de servidor de controle de versão, onde colocamos todo o código-fonte dos sistemas. Entretanto, apesar de existir um só fonte, foram criadas diversas versões, implantadas no decorrer do tempo (chamadas de artefatos, que são os arquivos binários JAR, WAR ou EAR).

Onde Bruno trabalha, não é diferente. Ele costumava guardar as diferentes versões do seu sistema em um diretório na rede, mas percebeu que controlar isso sozinho era insano e precisava de uma ferramenta para ajudar nesse controle.

Às vezes, existe a necessidade de voltar para uma versão específica, ou então, ter um controle rígido de qual versão está rodando em um ambiente de aceite e qual está em produção na nuvem. Talvez seja necessário pegar uma versão de um ano atrás.

Fazer esse controle manual não é recomendado, e é interessante termos um serviço para isso. A solução para esse problema chama-

se Artifactory.

O Artifactory é um sistema open source que é usado como repositório de arquivos binários. Existe uma versão paga dele, com suporte a outros tipos de repositório, como por exemplo, pacotes RPM de algumas distribuições Linux.

Ele tem outras vantagens também, como a de funcionar como um proxy dos artefatos externos, reunindo todas aquelas bibliotecas que o seu projeto usa em um único lugar, funcionando como um "cache" para a sua rede local.

Em um cenário sem o Artifactory, se algum projeto precisa, por exemplo, da biblioteca log4j, o Maven fará o download do endereço <http://central.maven.org/maven2/log4j/log4j/1.2.17/log4j-1.2.17.jar>. Se, na mesma equipe, vinte pessoas fizerem a mesma coisa, o Maven fará o mesmo download vinte vezes.

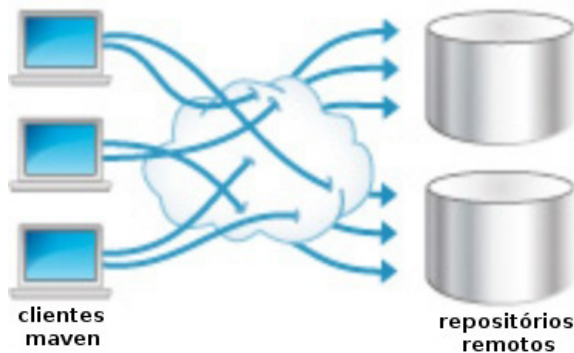


Figura 4.1: Cenário sem Artifactory

Em um cenário com o Artifactory, o download é feito do Artifactory, e ele, por sua vez, faz o download de <http://central.maven.org/maven2/log4j/log4j/1.2.17/log4j-1.2.17.jar> apenas uma vez.

Quando o restante da equipe precisar do arquivo do log4j, o download será feito do Artifactory, que fica na rede interna da empresa.



Figura 4.2: Cenário com Artifactory

Sua instalação está detalhada no Apêndice B.

4.2 INSTALANDO PLUGIN DO ARTIFACTORY

A integração com o Jenkins é feita por meio de um plugin criado pela JFrog, a mesma empresa que mantém o Artifactory.

Para instalar esse (e qualquer outro plugin), é necessário acessar a opção **Gerenciar Jenkins**, e depois **Gerenciar plugins**, conforme a figura:

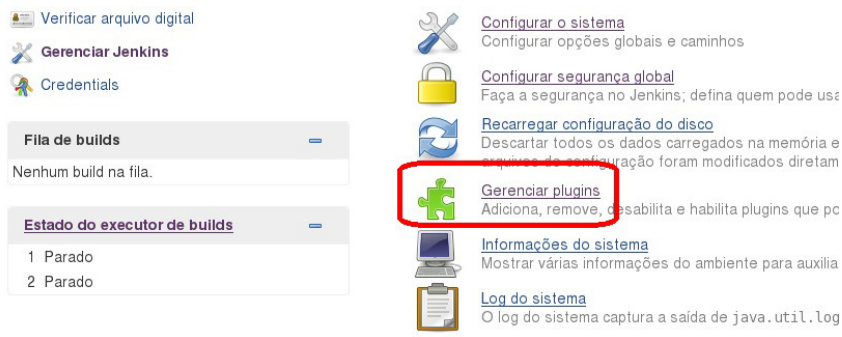


Figura 4.3: Instalar novo plugin

Em seguida, vá para abas disponíveis e, no campo de busca, digite `artifactory plugin`, conforme a figura:

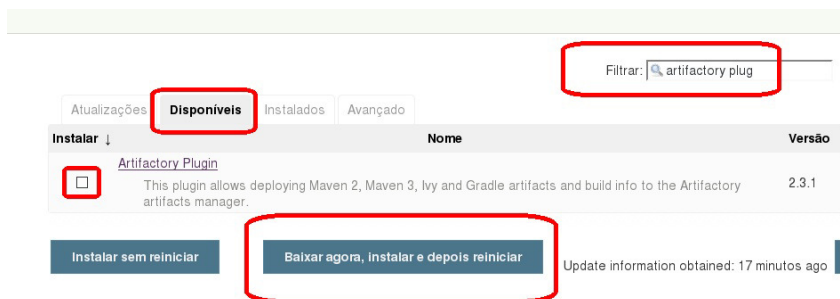


Figura 4.4: Instalar plugin do Artifactory

Depois, escolha o plugin e clique em `Baixar agora, instalar e depois reiniciar` para instalar. O Jenkins será reiniciado e o plugin instalado e ativado automaticamente.

Para conferir a instalação, visite novamente a opção de Gerenciar plugins, mas dessa vez consulte a aba `Instalados` e procure pelo plugin do Artifactory.

4.3 CONFIGURANDO O PLUGIN DO ARTIFACTORY

Acessando as configurações gerais (`Gerenciar Jenkins` -> `Configurar o sistema`), verificamos que foi criada uma nova opção chamada `Artifactory`. Precisamos adicionar o nosso servidor, informando sua URL e informações de usuário e senha, conforme a figura:

Artifactory

Artifactory servers

Artifactory URL

Default Deployer Credentials

User Name

Password

☐ Use Different Resolver Credentials

Avançado...

Test Connection

Figura 4.5: Configurando o servidor do Artifactory no plugin

Para testar o acesso, clique na opção **Test Connection**. Se der tudo certo, aparecerá a versão do Artifactory, como vemos a seguir:

Default Deployer Credentials

User Name

Password

☐ Use Different Resolver Credentials

Found Artifactory 3.8.0

Avançado...

Test Connection

Figura 4.6: Testando o acesso ao servidor do Artifactory

Para finalizar as alterações, clique em **salvar**.

4.4 PUBLICANDO PACOTES NO ARTIFACTORY

Vamos criar um job para publicar os artefatos do nosso projeto no Artifactory.

Felizmente, não precisamos configurar o SVN novamente em

um novo job. A opção mais fácil é copiar o job de um existente, no nosso caso, o `minhasmoedas-war`, como mostra a figura:

Nome do job

☐ **Construir um projeto de software free-style**
Esta é a central de funcionalidades do Jenkins. Ele construirá seu projeto, e você pode combinar qualquer SCM com qualquer sistema de builds, e ele até mesmo pode ser usado para outras jobs diferentes de builds de software.

☐ **Construir um projeto maven**
Construir um projeto maven. Jenkins tira vantagem de seus arquivos POM e reduz drasticamente a configuração. Ainda é um trabalho em progresso, mas disposto a aceitar feedback.

☐ **Construir projeto de múltiplas configurações**
Apropriado para projetos que necessitam de grande número de diferentes configurações, como teste em múltiplos ambientes, builds para plataformas específicas, etc.

☐ **External Job**
Este tipo de trabalho permite que você grave a execução de um processo rodando fora do Jenkins (talvez até de uma máquina remota.) Isto foi projetado para que você possa usar Jenkins como um painel principal de seus sistemas de automação. [Veja a documentação para mais detalhes.](#)

☒ **Copiar job existente**
Copiar de

Figura 4.7: Criar um job novo para o Artifactory

Depois de criarmos o job `minhasmoedas-artifactory`, precisamos ativar o plugin do Artifactory.

Em cada job, podemos adicionar ações antes e depois do build. Depois de um build, podemos adicionar uma publicação do artefato gerado no Artifactory. Fazemos isso escolhendo no combo Adicionar ação de pós-build a opção `Deploy artifacts to Artifactory`.

As opções padrões são suficientes para o nosso projeto, como a figura a seguir. Se os valores nos combos não vierem preenchidos, clique no botão de `refresh`.

Deploy artifacts to Artifactory

Artifactory server

Target releases repository
[Set a different value](#) [?](#)

Target snapshot repository
[Set a different value](#) [?](#)

Custom staging configuration [?](#)

Items refreshed successfully

[Refresh](#)

Figura 4.8: Criar um job novo para o Artifactory

Vamos apenas alterar as opções do Maven para ignorar os testes e fazer o build:

Construir

POM Raiz [?](#)

Metas e opções [?](#)

[Avançado...](#)

Figura 4.9: Opções do Maven

Para executar um build, dentro de Jenkins -> minhasmoedas-artifactory, clique em Construir agora.

4.5 VERIFICANDO O PACOTE PUBLICADO

Depois de o build terminar, o pacote será publicado no Artifactory.

No histórico de builds, aparece, além do número do build e sua data, um link para ver o artefato publicado, conforme ilustrado:



Figura 4.10: Artefato publicado no Jenkins

Clicando no link, podemos ver os detalhes do artefato direto no Artifactory:

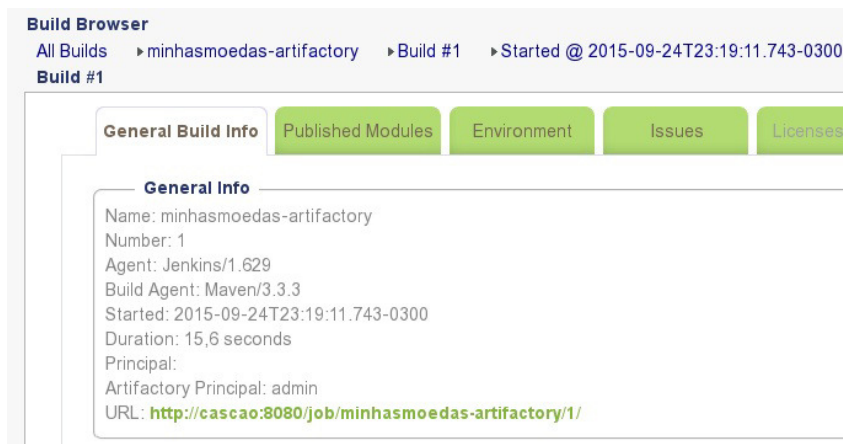


Figura 4.11: Artefato publicado no Artifactory

4.6 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Instalar o plugin do Artifactory;
- Configurar o plugin do Artifactory;
- Alterar um job para publicar no Artifactory.

Bruno tem um problema a menos. Ele se confundia com as versões publicadas, e agora tem um método confiável de organizar as suas versões, gerado pelo Jenkins e armazenado pelo Artifactory. No próximo capítulo, vamos aprender a colocar ordem e dependências nos jobs gerados.

CRIANDO UMA PIPELINE DE ENTREGAS

Boa comunicação é como uma plantinha que precisa ser regada todo dia. Luca Bastos

Depois de guardar os arquivos binários no Artifactory, certamente precisamos publicá-los em um ambiente de aceite e, depois, um em produção.

Bruno tem esse mesmo problema, e manualmente faz os deploys em cada ambiente. Vamos criar dois jobs separados para tratar de cada um.

5.1 PUBLICANDO EM ACEITE

De acordo com o nosso cenário, precisamos publicar no Tomcat existente em <http://chicobento:18080/>.

Existem várias opções de plugins para essa tarefa. Neste livro, vamos usar o `Deploy Plugin`.

A sua instalação é a mesma do capítulo anterior, porém no filtro colocamos `deploy to`, conforme a figura:



Figura 5.1: Instalando o Deploy Plugin

Depois de instalar o plugin, vamos criar um novo job chamado `minhasmoedas-aceite`, como cópia de `minhasmoedas-war`, da mesma maneira que fizemos no capítulo anterior.

Nas configurações do job criado, adicionamos as opções de build do Maven com `-DskipTests clean install`:

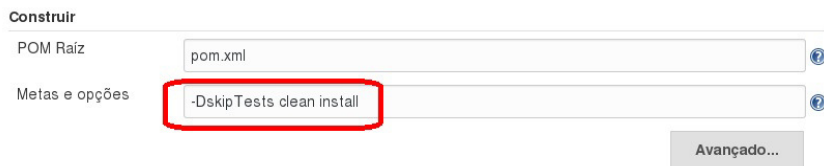


Figura 5.2: Opções de build do Maven

Em seguida, adicionamos a opção do `add-post-build step`, como vemos na figura:

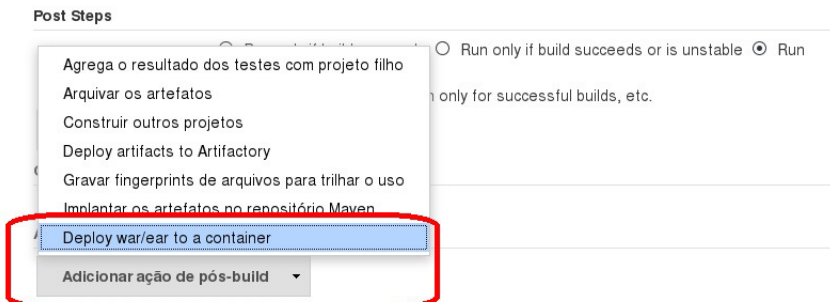
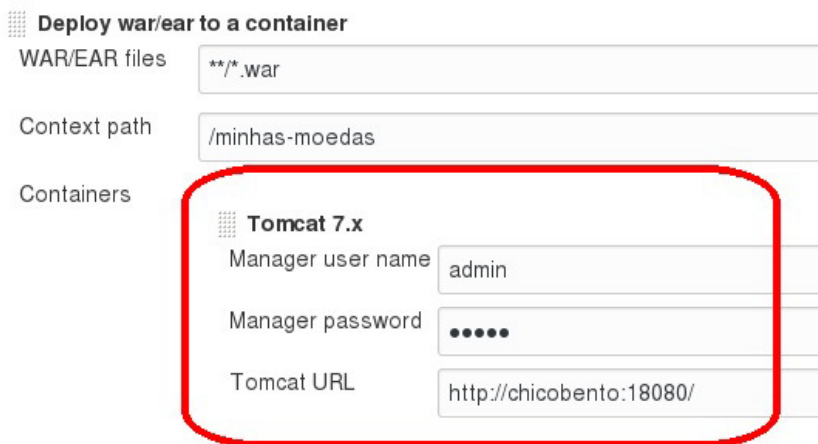


Figura 5.3: Adicionar a opção de deploy war

O plugin instalado fará o deploy do pacote via HTTP, por meio do console de administração do Tomcat. Portanto, precisamos da informação de usuário e senha, como mostra a figura a seguir. Precisamos também informar o contexto da aplicação `/minhas-moedas` e a sintaxe de envio dos arquivos `**/*.war`.



Deploy war/ear to a container

WAR/EAR files

Context path

Containers

Tomcat 7.x

Manager user name

Manager password

Tomcat URL

Figura 5.4: Informações de administração do Tomcat de aceite

Depois de salvar as alterações, iniciamos o build clicando em `construir agora` e, no final, o deploy é feito com sucesso, desde que o Tomcat esteja rodando. O resultado será semelhante ao da figura adiante:

```

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 21.094 s
[INFO] Finished at: 2015-10-04T17:38:41-03:00
[INFO] Final Memory: 55M/645M
[INFO] -----
[JENKINS] Archiving /var/lib/jenkins/home/jobs/minhasmoedas-aceite/workspace/
moedas/1.0.0/minhas-moedas-1.0.0.pom
[JENKINS] Archiving /var/lib/jenkins/home/jobs/minhasmoedas-aceite/workspace/
com.boaglio/minhas-moedas/1.0.0/minhas-moedas-1.0.0.war
channel stopped
Deploying /var/lib/jenkins/home/jobs/minhasmoedas-aceite/workspace/target
Tomcat 7.x Remote
[/var/lib/jenkins/home/jobs/minhasmoedas-aceite/workspace/target/minhas
a fresh deployment.
Deploying (/var/lib/jenkins/home/jobs/minhasmoedas-aceite/workspace/tar
Finished: SUCCESS

```

Figura 5.5: Resultado do build do deploy em aceite

Acessando o servidor, temos o resultado:

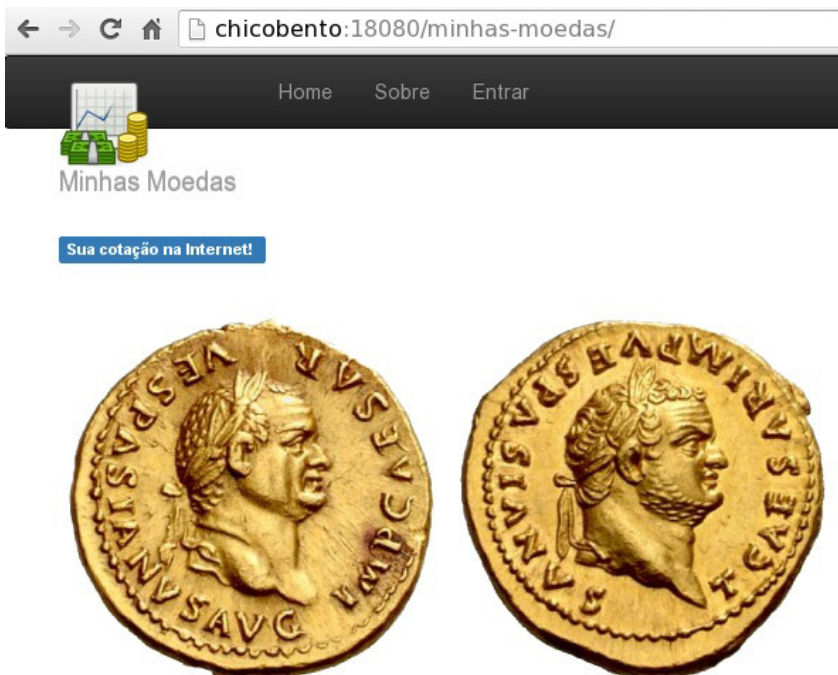


Figura 5.6: Aplicação Minhas Moedas em aceite

5.2 PUBLICANDO EM PRODUÇÃO

O nosso cenário de produção fica em <http://chicobento:28080/>, portanto, devemos publicar em outro servidor. A nossa tarefa agora ficou mais fácil, basta criar um job chamado `minhasmoedas-produção` como cópia do `minhasmoedas-aceite`.

Em seguida, alteramos as configurações para atualizar o servidor de destino, conforme a figura:

Deploy war/ear to a container

WAR/EAR files

Context path

Containers

Tomcat 7.x

Manager user name

Manager password

Tomcat URL

Excluir

Figura 5.7: Informações de administração do Tomcat de produção

Após fazermos o primeiro deploy, o resultado será semelhante. O deploy será feito no Tomcat de produção com sucesso, produzindo resultados semelhantes às figuras de aceite da seção anterior.

5.3 LIGANDO OS PONTOS

Até o momento, mesmo automatizando as tarefas de deploy, economizamos bastante tempo. Porém, ainda não temos nenhuma dependência entre os jobs.

Bruno conseguiu automatizar algumas tarefas, mas ainda não colocou nenhuma relação entre elas. Seria interessante existir uma sequência entre elas, por exemplo, após um build com sucesso,

guardar o artefato no Artifactory:

1. minhasmoedas-war
2. minhasmoedas-artifactory

O nome dessa cadeia de jobs em sequência é chamada de **pipeline**.

Cada job é chamado e, em caso de sucesso, chama o próximo. Esse comportamento pode ser alterado, e um próximo job pode ser chamado em caso de falha também.

Para informar que, após o build do `minhasmoedas-war`, será chamado o job `minhasmoedas-artifactory`, é preciso configurar no job uma ação de pós-build, conforme a figura:

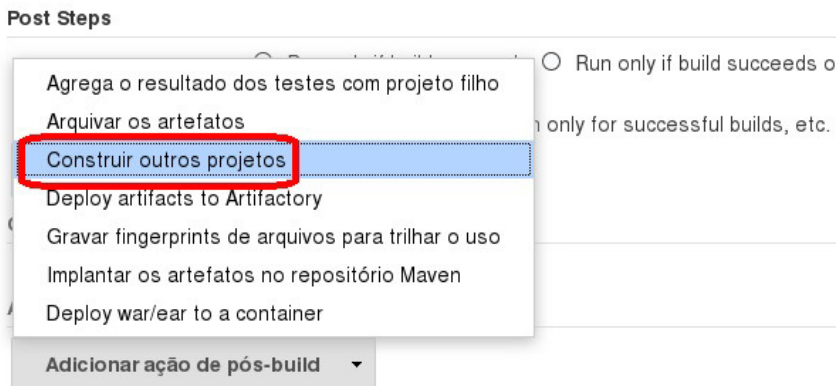


Figura 5.8: Adicionando uma ação de pós-build

Em seguida, na opção `Projetos para construir`, informamos o nome do job `minhasmoedas-artifactory`, como vemos na figura:



Figura 5.9: Escolhendo o job para chamar depois do build

O pipeline foi criado, mas não é possível visualizá-lo. Para isso, precisamos que seja instalado o **Delivery Pipeline Plugin**.

Depois de instalado, criaremos uma nova aba, clicando no ícone de somar conforme a figura adiante. As abas são usadas para agrupar ou classificar os diferentes jobs do Jenkins.



Figura 5.10: Criar uma nova aba de visualização

Depois, escolhemos o tipo **pipeline** :

Nome da view

☒ **Delivery Pipeline View**
Shows one or more delivery pipeline instances.

☐ **Ver lista**
Mostrar as jobs em uma lista simples. Você pode

Figura 5.11: Configuração da aba de visualização do pipeline

Depois de clicar em **OK**, o plugin oferece uma outra tela com várias opções. As únicas necessárias são o nome do pipeline e o do job inicial, como mostra a figura:

Pipelines

Components

Name ?

Initial Job ?

Final Job (optional)

Regular Expression

Figura 5.12: Escolhendo o job inicial para o pipeline

Depois de iniciar o build do job `minhasmoedas-war`, podemos visualizar o resultado do pipeline:



Figura 5.13: Visualização do pipeline simples

Temos até o momento um job que gera o pacote e outro que armazena no Artifactory, mas precisamos mesmo é automatizar o processo de publicação nos ambientes.

O nosso pipeline pode ser melhorado e adotar a seguinte sequência:

1. minhasmoedas-war
2. minhasmoedas-artifactory
3. minhasmoedas-aceite
4. minhasmoedas-produção

Configurando da mesma maneira demonstrada, temos um resultado semelhante à figura a seguir. Enquanto o Jenkins roda cada job, ele atualiza a visualização automaticamente.



Figura 5.14: Visualização do pipeline mais complexo

5.4 AGENDANDO BUILDS

Já que fizemos um job que faz backup dos códigos-fontes, não é interessante amarrá-lo em um pipeline, mas agendá-lo para rodar com uma certa frequência.

Vamos agendar para ele rodar todo dia às 23h. Isso é feito nas configurações, dentro da opção `Trigger de builds`, como mostra a figura mais adiante.

Podemos ter um job diário para publicar uma versão no ambiente de aceite, ou um job todo domingo para fazer um backup semanal. Para configurações de diferentes tempos, consulte a documentação do Jenkins, clicando no ícone de interrogação ao lado de cada campo.

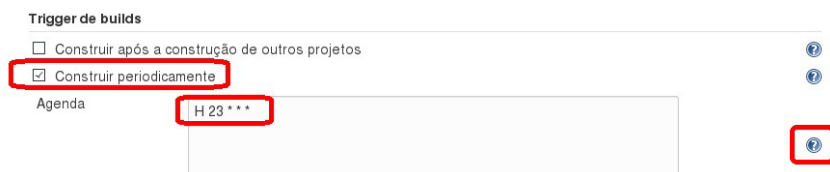


Figura 5.15: Agendando um build de um job

PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Instalar o plugin do deploy;
- Instalar o plugin do pipeline;
- Configurar o deploy em um Tomcat;
- Criar um pipeline de jobs;
- Criar uma aba de pipeline.

Bruno agora consegue fazer os deploys de maneira automática e, com o pipeline, em uma tacada só ele faz deploy em aceite e produção. Além disso, ele agendou o backup dos códigos-fontes para fazer toda noite, outra coisa a menos em sua lista de tarefas!

No próximo capítulo, vamos aprender a ativar a segurança do Jenkins para deixar o ambiente mais seguro.

AUTENTICAÇÃO E SEGURANÇA

Cuidado com os objetivos medíocres. Luca Bastos

Por padrão, o Jenkins vem totalmente liberado. Entretanto, é importante conhecer como restringir o seu acesso, desde o gerenciamento de plugins até visualizar a execução de algum job. Vamos verificar neste capítulo algumas opções que ele oferece.

Para acessar as configurações de segurança, acessamos a página de Gerenciar Jenkins , e depois o link Configurar segurança global :

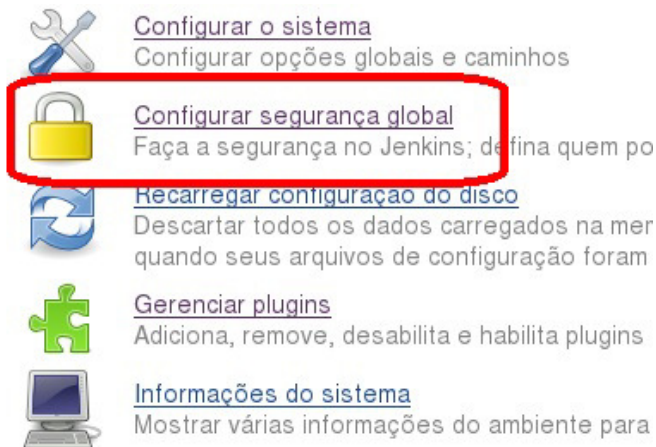
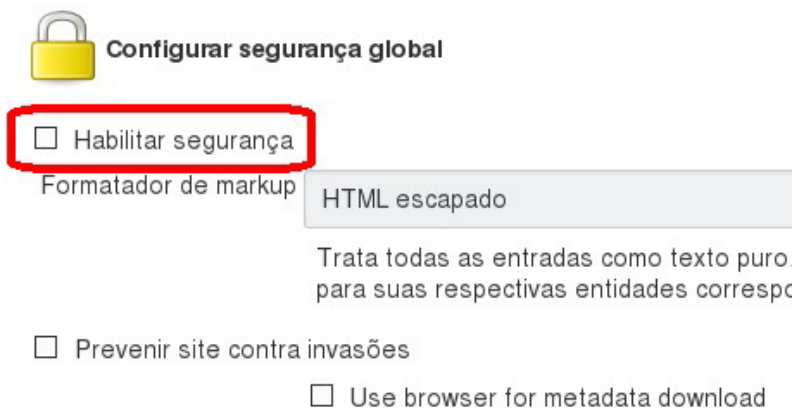


Figura 6.1: Acessar as configurações de segurança

Depois de acessar, ativamos o checkbox de segurança:



 **Configurar segurança global**

☐ **Habilitar segurança**

Formatador de markup: **HTML escapado**

Trata todas as entradas como texto puro para suas respectivas entidades correspondentes

☐ Prevenir site contra invasões

☐ Use browser for metadata download

Figura 6.2: Ativar a segurança

6.1 TIPOS DE AUTENTICAÇÃO

Existem diferentes maneiras de validar a entrada do usuário no Jenkins. São elas:

- **Base de dados interna do Jenkins:** cadastro interno do Jenkins;
- **Delegar para o servlet container:** usado em versões antigas do Jenkins, logo, é melhor não usar;
- **LDAP:** utiliza um Active Directory para fazer a autenticação do usuário;
- **Usuário ou grupo do UNIX:** usa os usuários do servidor UNIX onde o Jenkins está rodando para fazer a autenticação.

Para uma instalação simples, em que apenas desejamos criar um administrador e proteger o Jenkins, podemos usar o cadastro interno, ou usar o servidor local UNIX do Jenkins.

Em uma instalação dentro de uma empresa, na qual temos um

grupo de usuários razoável, é interessante delegar a autenticação para o servidor Active Directory, como mostra o exemplo da figura a seguir, com o domínio *boaglio.com* e usuário de rede *boaglio*.

The image shows the 'Habilitar segurança' (Enable security) section of the Jenkins configuration page. The 'Porta TCP para agentes slave JNLP' is set to 'Randômico'. Under 'Domínio (realm) de segurança', 'Active Directory' is selected. The 'Nome do domínio' is 'boaglio.com' and the 'Domain controller' is 'cascao.boaglio.com:3268'. The 'Bind DN' is 'boaglio' and the 'Bind Password' is masked with dots. The 'Group Membership Lookup Strategy' is set to 'Automatic'. At the bottom, there are radio buttons for authentication methods: 'Base de dados interna do Jenkins', 'Delegar para o container servlet', 'LDAP', 'Redmine User Auth', and 'Usuário Unix / grupo banco da dados'. The 'Autorização' (Authorization) section is partially visible at the bottom.

Figura 6.3: Configuração exemplo de Active Directory

6.2 TIPOS DE AUTORIZAÇÃO

Outra importante configuração é a autorização: definir o que um usuário anônimo (não logado) pode fazer, e o que um usuário logado pode fazer.

As opções existentes são:

- **Modo legado:** temos o usuário `admin` com controle total sobre o sistema, e os restantes com acesso apenas de leitura;
- **Qualquer um pode fazer qualquer coisa:** nenhuma

checagem de autorização é feita;

- **Segurança baseada em matriz:** controle detalhado em uma matriz de *usuários x privilégios*;
- **Project-based Matrix Authorization Strategy:** controle detalhado em uma matriz de *usuários x privilégios* por projeto;
- **Usuários logados conseguem fazer qualquer coisa:** opção usada para facilmente criar vários administradores.

Uma configuração simples e rápida é a da próxima figura, com o cadastro interno de Jenkins de vários administradores.

Controle de acesso

Domínio (realm) de segurança

- ☒ Base de dados interna do Jenkins
 - ☒ Permitir que os usuários se inscrevam
- ☐ Delegar para o servlet container
- ☐ LDAP
- ☐ Usuário ou grupo do UNIX

Autorização

- ☐ Modo legado
- ☐ Project-based Matrix Authorization Strategy
- ☐ Qualquer um pode fazer qualquer coisa
- ☐ Segurança baseada em matriz
- ☒ Usuários logados conseguem fazer qualquer coisa

Figura 6.4: Configuração exemplo de segurança

6.3 RECUPERANDO O ACESSO PERDIDO

Não é muito difícil esquecer uma senha e perder o acesso de administração do Jenkins. Tendo acesso ao servidor, conseguimos acessar o `JENKINS_HOME` e, como já vimos no primeiro capítulo, todas as configurações ficam lá, já que o Jenkins não usa banco de dados, inclusive as de segurança.

Dentro do diretório do `JENKINS_HOME` , existe o arquivo `config.xml` , que possui as principais configurações do Jenkins. Se for necessário remover a autenticação, o procedimento é:

1. Derrube o serviço do Jenkins;
2. Abra o arquivo `JENKINS_HOME/config.xml` em um editor de textos;
3. Altere a tag `useSecurity` de `true` para `false` ;
4. Remova a tag `securityRealm` e tudo dentro dela;
5. Remova a tag `authorizationStrategy` e tudo dentro dela;
6. Suba o serviço do Jenkins.

Com isso, o Jenkins volta a ser liberado, mas todas as configurações de autenticação e autorização foram removidas (e não apenas desligadas).

6.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- Como ativar a autenticação no Jenkins;
- Os diferentes tipos de autorização;
- Como recuperar o acesso.

Depois que Bruno instalou o Jenkins em sua empresa, ele pode escolher qual a melhor maneira de manter as suas configurações do Jenkins e seus deploys mais seguros.

No próximo capítulo, vamos aprender a atualizar o nosso banco de dados com o Jenkins.

VALIDANDO E ATUALIZANDO O BANCO DE DADOS

Não ser somente uma comunidade de profissionais de TI, é preciso ver sob o ponto de vista de todos. Luca Bastos

Bruno já automatizou o deploy, mas continua atualizando o banco de dados manualmente. Assim, é preciso resolver esse problema, pois rodar um script de banco errado certamente causará um erro em produção, e sendo essa etapa um processo manual, existe uma chance razoável de isso acontecer.

Vamos mostrar um exemplo de atualizar o banco de dados, mas, antes disso, precisamos validar o horário de execução dessa atualização. Para realizar essas tarefas, será necessária a instalação do Gradle (que é uma ferramenta de automação de builds). Saiba mais sobre ele no Apêndice F.

7.1 INSTALANDO OS PLUGINS

Para executar as duas tarefas de validar o horário e atualizar o banco de dados, precisamos instalar dois plugins: inicialmente o Gradle plugin , para usar com o Flyway, conforme a figura:



Figura 7.1: Instalando o Gradle plugin

Em seguida, o **Groovy Plugin**, para usar um script de validar o horário:

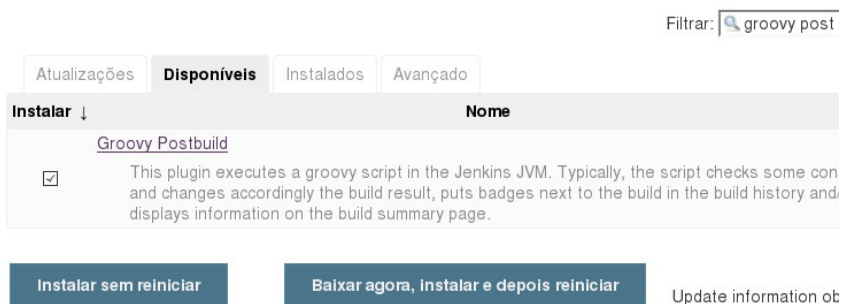


Figura 7.2: Instalando o Groovy Postbuild plugin

7.2 USANDO O SCRIPT DE VALIDAR O HORÁRIO

Da maneira como está configurado o job **minhasmoedas-produção**, para executar uma alteração em produção, basta agendar um build.

O problema é que, se isso acontecer no horário comercial, o site ficará indisponível por alguns instantes, algo que não desejamos. Uma saída é fazer uma validação via script, que verifique o horário

atual e aborte o build se necessário.

Vamos usar uma característica interessante do Groovy Postbuild , que é chamar dentro do script subrotinas do Jenkins, como `buildFailure()` ou `buildSuccess()` . O script usado será este a seguir, vamos ver em partes.

Aqui, chamamos a rotina `manager.createSummary` para exibir um ícone e uma mensagem no painel do Jenkins.

```
manager.createSummary("gear2.gif")
    .appendText("<h2>Verificando horário...</h2>", false)
```

Depois, definimos a variável `now` para determinar a hora atual:

```
def now = new Date()
def hora = now.getHours()
println "Horário agora: "+hora
```

Finalmente, validamos o horário: se for entre meia-noite e sete da manhã, será exibida uma mensagem de OK:

```
if (hora>0 && hora< 7) {
    manager.createSummary("green.gif")
    .appendText("<h1>Horário OK para deploy!</h1>",
        false, false, false, "green")
}
```

Em caso negativo, será exibida uma mensagem de que o deploy foi cancelado.

```
else {
    manager.createSummary("warning.gif")
    .appendText("<h1>
        Horário fora da janela, deploy cancelado!
    </h1>",
        false, false, false, "red")
    manager.buildFailure()
}
```

Vamos editar as configurações do job `minhasmoedas-produção` , e escolher e adicionar uma ação pós-build de Groovy Postbuild , conforme a figura a seguir:

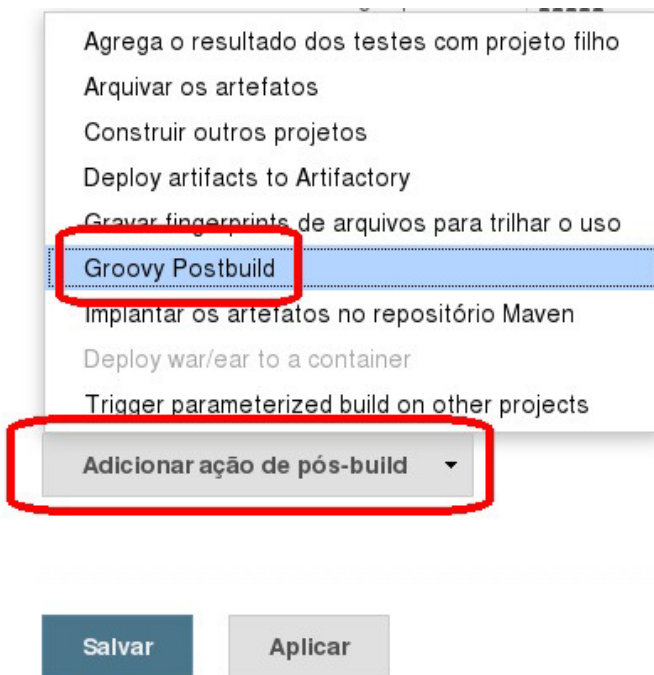


Figura 7.3: Adicionando ação do Groovy Postbuild

Em seguida, adicionamos o script:

Ações de pós-build

Groovy Postbuild

Groovy Script

```
manager.createSummary("gear2.gif").appendText("<h2>Verificando horário...</h2>", false)

def now = new Date()
def hora = now.getHours()

println "Horário agora: "+hora
if (hora>0 && hora< 7) {
    manager.createSummary("green.gif").appendText("<h1>Horário OK para deploy!</h1>",
false, false, false, "green")
}
else {
    manager.createSummary("warning.gif").appendText("<h1>Horário fora da janela, deploy
cancelado!</h1>", false, false, false, "red")
    manager.buildFailure()
}
```

☐ Use Groovy Sandbox

Additional classpath

Add entry

If the script fails:

Mark build as failed

Figura 7.4: Configurando o Groovy Postbuild

Ao executar o Job em um horário permitido, será exibida a figura:

Jenkins > minhasmoedas-produção > #5

[Voltar ao projeto](#)

[Estado pessoal](#)

[Alterações](#)

[Saída do console](#)

[Editar informações de compilação](#)

[Excluir este build](#)

[Tag this build](#)

[Reimplantar artefatos](#)

[Ver fingerprints](#)

[Build anterior](#)

**Build #5 (12/10/2015 20:10:34)**

**Revision: 20**
Sem mudanças.

**Iniciado pelo usuário anônimo**

**Verificando horário...**

**Horário OK para deploy!**

Figura 7.5: Build em horário permitido

Caso contrário, será exibida a tela a seguir:



Figura 7.6: Build em horário não permitido

7.3 FLYWAY

Vamos usar o Flyway como task do Gradle. Para isso, é preciso adicionar uma instalação válida do Gradle, ou pedir para o Jenkins baixar e instalar.

Para essa configuração, acessamos a opção de Configurar o sistema, e configuramos as opções do Gradle como vemos na figura a seguir:

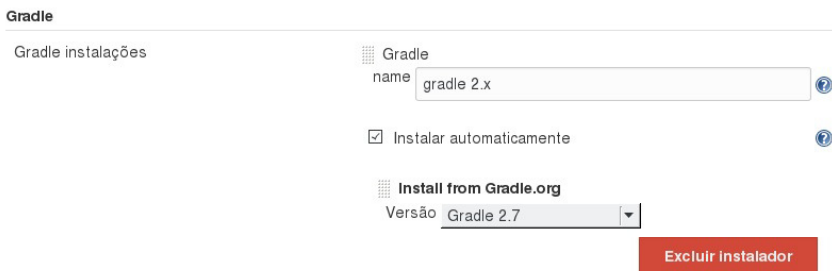


Figura 7.7: Configurações do plugin do Gradle

Tudo certo, os plugins estão configurados e prontos para serem usados nos jobs.

7.4 ATUALIZANDO A BASE DE DADOS DE ACEITE

Vamos criar um job chamado `minhasmoedas-atualiza-banco-aceite` com opção de Construir um projeto de software free-style , como a figura:

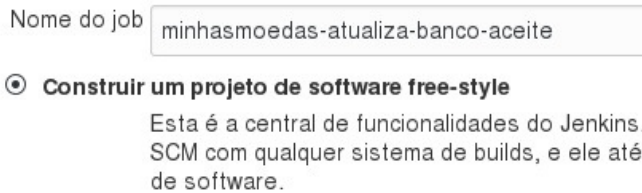


Figura 7.8: Construindo um job freestyle

Configure na opção de SVN com a URL para o checkout dos arquivos, exatamente como está nos outros jobs (se precisar consultar de novo, reveja a seção *Pacote Web*, no capítulo *Builds*).

Em seguida, vamos configurar a chamada do *Gradle script*, escolhendo do combo a versão instalada do Gradle (2.x) e colocando os parâmetros (*switches*) a seguir, que apontam para a base de aceite:

```
-x compileJava
-x testClasses
-x compileTestJava
-Pflyway.user=minhasmoedas_aceite
-Pflyway.password=aceite
-Pflyway.url="jdbc:oracle:thin:@cascao:1521:O12C"
```

Em seguida, informamos as tasks do Gradle (no nosso caso, a `flywayMigrate`) para executar os scripts de atualização do banco de dados.

```
clean
processResources
flywayMigrate
```

As configurações serão semelhantes à figura:

Invoke Gradle script

☒ Invoke Gradle
Gradle Version: gradle 2.x

☐ Use Gradle Wrapper
Build step description:

Switches: -x compileJava -x testClasses -x compileTestJava -Pflyway.user=minhasr

Tasks: clean processResources flywayMigrate

Root Build script:

Build File:

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

Force GRADLE_USER_HOME to use workspace ☐

Figura 7.9: Configurações do Gradle no job

Supondo uma base de aceite limpa, ao executar o build, o Flyway roda os scripts e atualiza para a versão atual (no nosso caso, a versão 2), conforme a figura a seguir:

```
Executing task ':flywayMigrate' (up-to-date check took 0.0 secs) due to:
  Task has not declared any outputs.
Flyway 3.2.1 by Boxfuse
Database: jdbc:oracle:thin:@cascao:1521:012C (Oracle 12.1)
Validated 2 migrations (execution time 00:00.013s)
Creating Metadata table: "MINHASMOEDAS_ACEITE" "schema_version"
Current version of schema "MINHASMOEDAS_ACEITE": << Empty Schema >>
Migrating schema "MINHASMOEDAS_ACEITE" to version 1 - Cria tabela usuario
Migrating schema "MINHASMOEDAS_ACEITE" to version 2 - Cadastra usuarios
Successfully applied 2 migrations to schema "MINHASMOEDAS_ACEITE" (execution
:flywayMigrate (Thread[main,5,main]) completed. Took 1.369 secs.

BUILD SUCCESSFUL

Total time: 4.77 secs
Stopped 0 compiler daemon(s).

This build could be faster, please consider using the Gradle Daemon: https://gradle.daemon.html
Build step 'Invoke Gradle script' changed build result to SUCCESS
Finished: SUCCESS
```

Figura 7.10: Resultado da base de aceite migrada com sucesso

7.5 ATUALIZANDO A BASE DE DADOS DE PRODUÇÃO

Configurar o job de atualizar a base de produção é bem simples:

1. Criar um job `minhasmoedas-atualiza-banco-prod` como cópia de `minhasmoedas-atualiza-banco-aceite` ;
2. Atualizar os parâmetros do Gradle script para apontar para a base de produção.

Depois de executar o job, o resultado esperado será o da figura adiante:

```
:flywayMigrate (Thread[main,5,main]) started.
:flywayMigrate
Executing task ':flywayMigrate' (up-to-date check took 0.001 secs) due to:
  task has not declared any outputs.
Flyway 3.2.1 by Boxfuse
Database: jdbc:oracle:thin:@cascao:1521:012C (Oracle 12.1)
Validated 2 migrations (execution time 00:00.011s)
Creating Metadata table: "MINHASMOEDAS_PROD"."schema_version"
Current version of schema "MINHASMOEDAS_PROD": << Empty Schema >>
Migrating schema "MINHASMOEDAS_PROD" to version 1 - Cria tabela usuario
Migrating schema "MINHASMOEDAS_PROD" to version 2 - Cadastra usuarios
Successfully applied 2 migrations to schema "MINHASMOEDAS_PROD" (execution
:flywayMigrate (Thread[main,5,main]) completed. Took 1.363 secs.

BUILD SUCCESSFUL

Total time: 4.198 secs
Stopped 0 compiler daemon(s).
```

Figura 7.11: Resultado da base de produção migrada com sucesso

7.6 ATUALIZANDO A PIPELINE

Vamos aumentar o nosso pipeline atual para rodar a atualização do banco de dados *antes* de publicar no Tomcat.

Consultando o capítulo *Criando uma pipeline de entregas*, verificamos que o cenário atual está assim:

1. `minhasmoedas-war`
2. `minhasmoedas-artifactory`
3. `minhasmoedas-aceite`
4. `minhasmoedas-produção`

O que faremos é aumentar para:

1. minhasmoedas-war
2. minhasmoedas-artifactory
3. ::minhasmoedas-atualiza-banco-aceite::
4. minhasmoedas-aceite
5. ::minhasmoedas-atualiza-banco-prod::
6. minhasmoedas-produção

Ao iniciarmos o job `minhasmoedas-war`, será iniciada todo o pipeline de jobs, e podemos acompanhar a sua execução na visualização de pipeline, que, por sua vez, possui uma opção *fullscreen* que fica mais fácil de ver, conforme a figura:

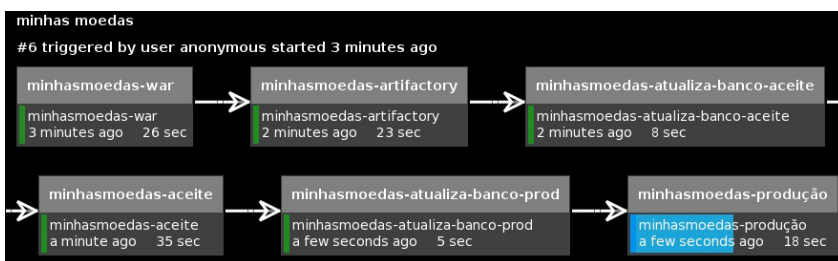


Figura 7.12: Pipeline atualizado em fullscreen

7.7 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Instalar os plugins de Groovy e Gradle;
- Configurar o plugin de Groovy para rodar um script;
- Configurar o plugin de Gradle para rodar uma task;
- Atualizar a base de dados de diferentes ambientes com o Flyway.

Depois que Bruno configurou o Groovy no Jenkins, ele colocou um script que validará o horário de deploy em produção para evitar

surpresas desagradáveis. Além disso, ele automatizou também a atualização de scripts nos bancos de dados para o Flyway, para não se preocupar mais com pacote subindo no Tomcat sem ter a base atualizada antes.

No próximo capítulo, vamos aprender como melhorar a qualidade de código do nosso sistema usando o Sonar.

AUMENTANDO A QUALIDADE DAS ENTREGAS

Não somente uma parceria produtiva com o cliente, é preciso ter o cliente sempre disposto a validar tudo. Luca Bastos

Não existe nada pior do que descobrir um problema de programação no ambiente de produção, principalmente se ele era bem simples e poderia ter sido evitado com uma boa prática de programação.

Bruno teve um problema parecido semana passada. O recém-contratado, que se dizia Java sênior, commitou uma pérola no código e não testou, o que gerou um erro de `NullPointerException` na cara do cliente, fazendo a empresa perder várias vendas nesse dia.

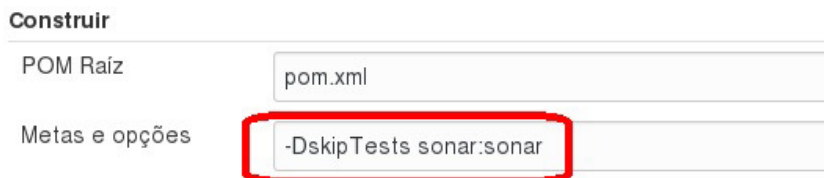
Uma opção interessante para evitarmos erros assim é criarmos um job para analisar o código-fonte antes de ele ser publicado. A ferramenta que nos ajudará nisso é o Sonar ou SonarQube, que veremos mais no Apêndice D.

8.1 TESTANDO O SOFTWARE ANTES

Vamos criar um job para rodar essa análise e alimentar o Sonar.

Para facilitar o nosso trabalho, vamos criar um novo job `minhasmoedas-sonar`, como cópia do `minhasmoedas-war`.

Em seguida, alteramos as configurações do Maven, conforme a figura:



Construir

POM Raiz

Metas e opções

Figura 8.1: Configurações do Sonar no Maven

Como copiamos o job, copiamos também a configuração do pipeline para chamar outro job. Portanto, precisamos excluir essa opção, como mostra a figura a seguir:



Ações de pós-build

Construir outros projetos

Projetos para construir

☒ Disparar apenas se o build estiver estável

☐ Disparar mesmo se o build estiver instável

☐ Disparar mesmo se o build falhar

Excluir

Adicionar ação de pós-build ▼

Figura 8.2: Removendo ação de pós-build

8.2 GERANDO AS MÉTRICAS NO SONAR

Depois de gerarmos o build, observamos que as métricas foram geradas e, no console do Jenkins, foi fornecido um link de acesso:

```
[INFO] [15:41:15.077] Store results in database
[INFO] [15:41:15.434] Analysis reports generated in 21ms, dir size=4 KB
[INFO] [15:41:15.441] Analysis reports compressed in 7ms, zip size=6 KB
[INFO] [15:41:15.502] Analysis reports sent to server in 0ms
[INFO] [15:41:15.503] ANALYSIS SUCCESSFUL, you can browse http://cascao:9000/dashboard/index/com.boaglio:minhas-moedas
[INFO] [15:41:15.503] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.630 s
[INFO] Finished at: 2015-10-18T15:41:16-02:00
[INFO] Final Memory: 31M/578M
```

Figura 8.3: Build gerando métricas do Sonar

Ao acessarmos o link mencionado, conseguimos acessar o Sonar e analisar suas métricas fornecidas.

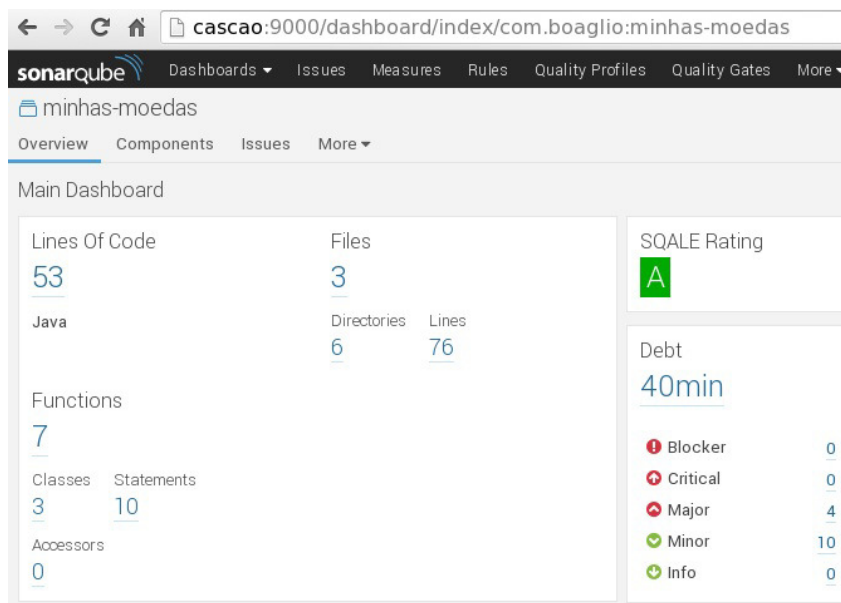


Figura 8.4: Resultados das métricas no Sonar

8.3 ATUALIZANDO A PIPELINE

Vamos atualizar o pipeline para que, depois do job `minhasmoedas-war`, que, além de gerar um pacote WAR e chamar o job `minhasmoedas-artifactory`, chame também o job do

minhasmoedas-sonar , como mostra a figura:

Ações de pós-build

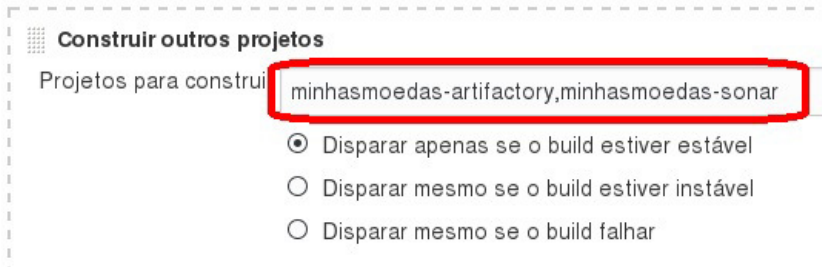


Figura 8.5: Atualizando o pipeline com a chamada do Sonar

Ao iniciar a execução do novo pipeline, será exibido um diagrama semelhante ao seguinte:

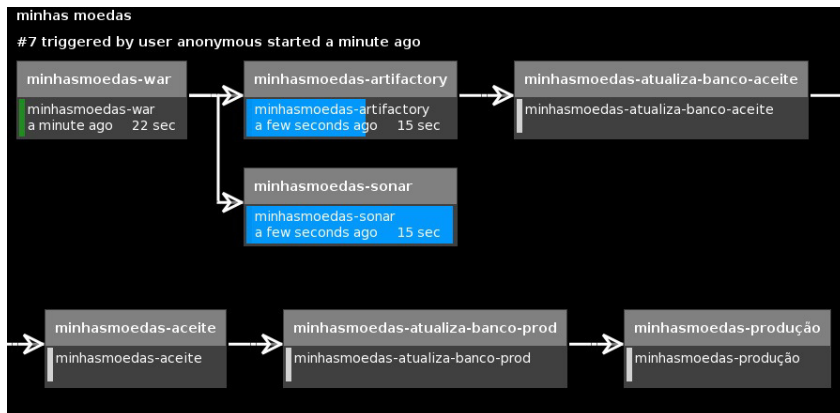


Figura 8.6: Pipeline com o Sonar em execução

8.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Configurar o Sonar no Jenkins;
- Atualizar o pipeline.

Depois que Bruno configurou o Sonar, ele consegue ver as métricas nele e evitar aquele `NullPointerException` na cara do usuário. Com os relatórios do Sonar, ele consegue melhorar o seu código e evitar um erro em produção, já que nos relatórios são apontados os problemas e também sugeridas soluções.

No próximo capítulo, vamos aprender como usar o plugin de promoção do Jenkins, para permitir (ou não) que os jobs sejam executados.

PROMOVENDO SUAS ENTREGAS

O importante na vida é se emocionar! Luca Bastos

Em um ambiente corporativo, podemos usar o Jenkins para automatizar várias tarefas, entre elas o deploy em produção. Entretanto, a execução dessa tarefa não pode ser liberada para qualquer pessoa.

Bruno já teve problema de erro em produção em sua equipe, e não quer que isso se repita. Portanto, ele não pode deixar que qualquer um faça o deploy em produção. Ele precisa que somente seu chefe, que não sabe nada de Jenkins, consiga executar esse job específico.

Para esse cenário, existe um plugin que permite que apenas algumas pessoas possam executar, ou melhor, aprovar a execução do build.

Usando o Promoted Builds Plugin , conseguimos restrições interessantes, como veremos adiante.

9.1 INSTALANDO O PLUGIN

Acessando a tela de Gerenciar plugins , colocamos no filtro promoted , como vemos na figura:

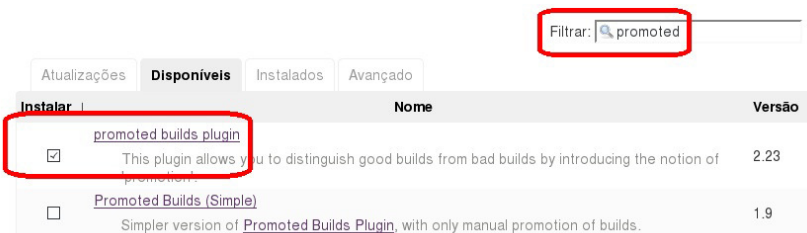


Figura 9.1: Instalando o Promoted Builds Plugin

9.2 PROMOVENDO UM JOB

Depois de instalado, podemos ativar a opção de promover um build de um job nas configurações do `minhasmoedas-war`, logo abaixo da descrição ativando o checkbox `Promote builds when...`.

Nela fornecemos o nome dessa aprovação (`deploy` no Artifactory), o ícone que aparecerá no painel do Jenkins (`Gold star`), e a lista de usuários que podem aprovar os builds (`admin`), como mostra a figura:

☒ Promote builds when...

Promotion process

Name	deploy no Artifactory
Icon	Gold star

☐ Restrict where this promotion process can be run

Criteria

☒ Only when manually approved	
Approvers	admin

Approval Parameters

Add Parameter ▼

☐ Promote immediately once the build is complete

☐ Promote immediately once the build is complete based on build parameters

☐ When the following downstream projects build successfully

☐ When the following upstream promotions are promoted

Figura 9.2: Configurando o Promoted Builds Plugin – parte 1

Em seguida, após o build promovido (ou aprovado), chamaremos o job do Artifactory para continuar o pipeline criado anteriormente:

Actions

 Construir outros projetos	
Projetos para construir	minhasmoedas-artifactory

☒ Disparar apenas se o build estiver estável

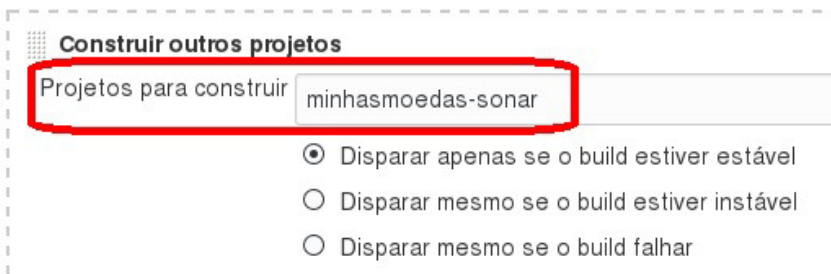
☐ Disparar mesmo se o build estiver instável

☐ Disparar mesmo se o build falhar

Figura 9.3: Configurando o Promoted Builds Plugin – parte 2

Depois, devemos atualizar a opção de ações de pós-build e remover a chamada do minhasmoedas-artifactory, deixando apenas chamando o job do Sonar:

Ações de pós-build



Construir outros projetos

Projetos para construir

☒ Disparar apenas se o build estiver estável

☐ Disparar mesmo se o build estiver instável

☐ Disparar mesmo se o build falhar

Figura 9.4: Configurando o Promoted Builds Plugin - parte 3

Com essa configuração, temos o nosso job protegido e apenas o usuário `admin` poderá promover os builds desse job.

9.3 ACESSANDO O JOB SEM PERMISSÃO

Quando um usuário sem permissão acessa o painel, ele consegue ver que o build foi iniciado e acessar a tela de promoção, clicando em **Promotion Status** :



[Voltar ao projeto](#)

[Estado pessoal](#)

[Alterações](#)

[Saída do console](#)

[View Build Information](#)

[Promotion Status](#)

[Ver fingerprints](#)

[Build anterior](#)

Build #10 (19/10/2015 02:33:48)

Revision: 20
Sem mudanças.

Iniciado pelo(a) usuário(a) [admin](#)

Módulos de construção

[minhas-moedas](#) 11 segundos

Figura 9.5: Acessando promoção de builds sem permissão

Depois, conseguimos apenas visualizar quem pode aprovar esse build, mas não temos permissão para fazer nada.

Promotions



deploy no Artifactory

This promotion has not happened.

Met Qualification

Unmet Qualification

Manual Approval

Approvers

admin

List of users or groups that can approve this promotion

Figura 9.6: Acessando promoção de builds sem permissão

9.4 ACESSANDO O JOB COM PERMISSÃO

Depois de logar no Jenkins como `admin`, essa mesma tela de promoção aparece o botão `Approve`:

Promotions

★ deploy no Artifactory

This promotion has not happened.

Force promotion

Met Qualification

Unmet Qualification

Manual Approval



Approvers

List of users or groups that can approve this promotion

Approve

Figura 9.7: Aprovando o build

Após aprovar, o build é executado. Para ilustrar a promoção, o ícone é alterado para a imagem selecionada na configuração (gold star):

Promotions

★ deploy no Artifactory

Promotion History

Figura 9.8: Acessando o build promovido

9.5 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Instalar e configurar o plugin de promotions;
- Configurar um job para promover os seus builds.

Depois que Bruno instalou o Promoted Builds Plugin, os problemas de deploy automático por qualquer um acabaram, apenas algumas pessoas têm esse privilégio. E isso não impediu o resto da equipe de ainda usar o Jenkins para acompanhar os deploys nos diferentes ambientes existentes.

No próximo capítulo, vamos aprender a fazer um teste completo na nossa aplicação web com o Jenkins.

TESTANDO SUA APLICAÇÃO

Nossa aplicação está na rede e pode ser acessada por diferentes browsers.

Bruno precisa testar sua aplicação também. Como ele poderia automatizar isso? Vamos ver adiante como automatizar um teste de aplicação no browser, dentro e fora do servidor do Jenkins.

10.1 INSTALANDO O PLUGIN

Acessando a tela de Gerenciar plugins , colocamos no filtro `xvfb` , conforme a figura:



Figura 10.1: Instalar o XVFB plugin

Com esse plugin, temos o ambiente gráfico que o Selenium

precisa para chamar o browser e navegar no sistema.

Para ativar o plugin, precisamos configurar as opções gerais do Jenkins, dando um nome, e ativar a instalação automática:

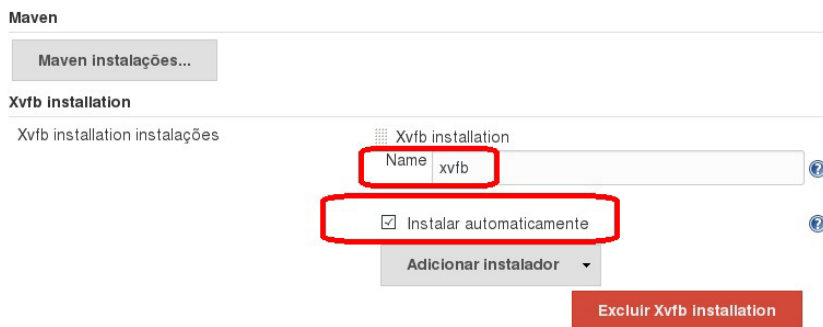


Figura 10.2: Ajustar para instalar automaticamente

10.2 ATIVANDO O JOB PARA USAR O SELENIUM

Ao editar o job `minhasmoedas-war`, ativamos a opção de `Start Xvfb before the build, and shut it down after` e colocamos as opções conforme a figura:

☒ Start Xvfb before the build, and shut it down after.

Xvfb installation xvfb

Let Xvfb choose display name ☒

Xvfb specific display name

Start only on nodes labeled

I'm running this job in parallel on same node ☐

Timeout in seconds

Xvfb screen

Xvfb display name offset

Xvfb additional options

Log Xvfb output ☒

Shutdown Xvfb with whole job, not just with the main build action ☐

☐ Create Delivery Pipeline version

☐ Enable Artifactory release management

☐ Resolve artifacts from Artifactory

Figura 10.3: Configurar o Xvfb para o job

Nas metas e opções do Maven, podemos alterar para `clean install`. Dessa maneira, o próximo build rodará todos os testes, inclusive os do Selenium.

10.3 SELENIUM

Ao executar o job, fazemos dois testes simples: o primeiro acessa a homepage e verifica se o texto *Minhas Moedas* existe. O segundo teste é mais complexo, ele faz o login no sistema e valida a existência de um trecho do texto *Cotação de hoje*.

Iniciamos abrindo a página de login com o Firefox:

```
WebDriver driver = new FirefoxDriver();
driver.get("http://chicobento:18080/minhas-moedas/entrar");
```

Depois, navegamos informando o usuário e a senha, e clicamos no botão de `submit`.

```
driver.findElement(By.id("username")).sendKeys("boaglio");
driver.findElement(By.id("password")).sendKeys("boaglio");
driver.findElement(By.className("btn")).submit();
```

E, finalmente, testamos se existe o texto de hoje .

```
if (driver.getPageSource().contains("de hoje")) {
    System.out.println("login ok !");
}
driver.quit();
```

O projeto `minhasmoedas` utiliza opções interessantes do Selenium para, além de fazer o teste, gerar screenshots e escolher via parâmetro qual browser usar: o Firefox ou Internet Explorer.

Executando o build, depois de rodar o browser, podemos verificar os screenshots no workspace do projeto:

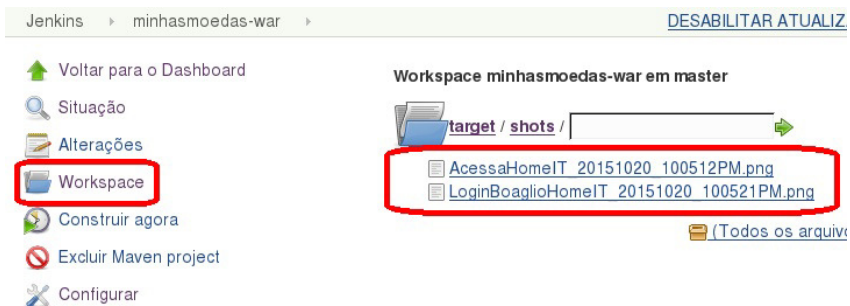


Figura 10.4: Screenshots no workspace

10.4 TESTANDO FORA DO SERVIDOR COM O JENKINS SLAVE

Agora, precisamos fazer testes no Internet Explorer, mas isso não é possível fazer nativamente no Linux.

Para esse tipo de job, precisamos usar outra máquina que rodará o teste gerenciado pelo Jenkins. Isso é chamado de um *slave* (escravo).

Cadastrando um novo desktop como slave

Para cadastrar uma outra máquina (desktop ou servidor) como slave, acessamos a tela de Gerenciar Jenkins , e depois a opção de Gerenciar nós . Então, clicamos na opção de Novo nó , como mostra a figura:

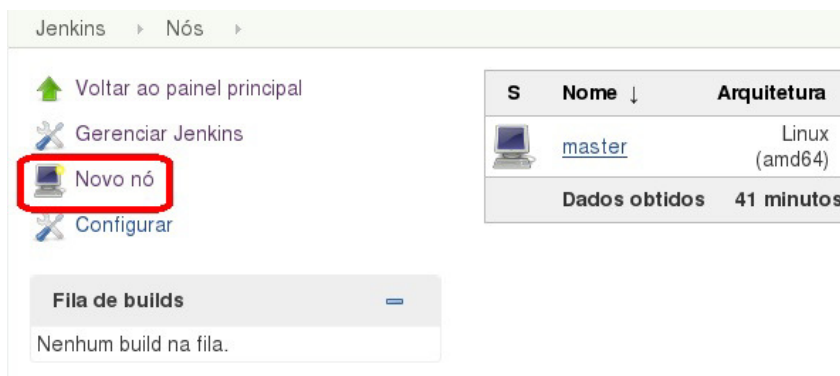


Figura 10.5: Link para adicionar um novo nó

Depois informamos o nome do nó e escolhemos a única opção existente, de criar um slave burro:

Nome do nó

☒ **Slave burro**

Adiciona um slave simples e burro ao Jenkins. É chamado "burro" porque o Jenkins não provê um maior nível de integração com estes slaves, como um provisionamento dinâmico. Selecione este tipo se nenhum outro slave se aplica; por exemplo como quando você está adicionando um computador físico, máquinas virtuais gerenciáveis fora do Jenkins, etc.

Figura 10.6: Cadastrar o slave – parte 1

Informe um nome (desktop), uma descrição (testes no IE), o diretório root remoto (`C:\jenkins`) e o método de lançamento (lançar os agentes slave via JNLP), como na figura:

Nome	desktop
Descrição	testes no IE
Número de executores	1
Diretório root remoto	C:\jenkins
Rótulos	
Uso	Utilize este slave, tanto quanto possível
Método de lançamento	Lançar os agentes slave via JNLP
Disponibilidade	Manter este slave ligado tanto quanto for possível

Figura 10.7: Cadastrar o slave – parte 2

Além disso, vamos especificar detalhes da instalação desse desktop, como a versão do JDK já instalado e o JAVA_HOME :

Propriedades dos nós

☒ Ferramentas locais

Lista dos locais das ferramentas

Nome	(JDK) 1.8
Início	C:\JDK

Adicionar

☒ Variáveis de ambiente

Lista de pares de chave-valor

nome	JAVA_HOME
valor	C:\JDK

Adicionar

Figura 10.8: Cadastrar o slave – parte 3

Depois, a parte final é feita a partir do desktop, logando-se como usuário administrador da máquina. Acessando o Jenkins via browser no link (<http://cascao:8080/computer/desktop/>), clicamos no botão **Launch** , ou executamos a linha de comando, como mostra a figura:



Figura 10.9: Registrar o slave no Jenkins – parte 1

Precisamos confirmar a opção de segurança para registrar o slave:

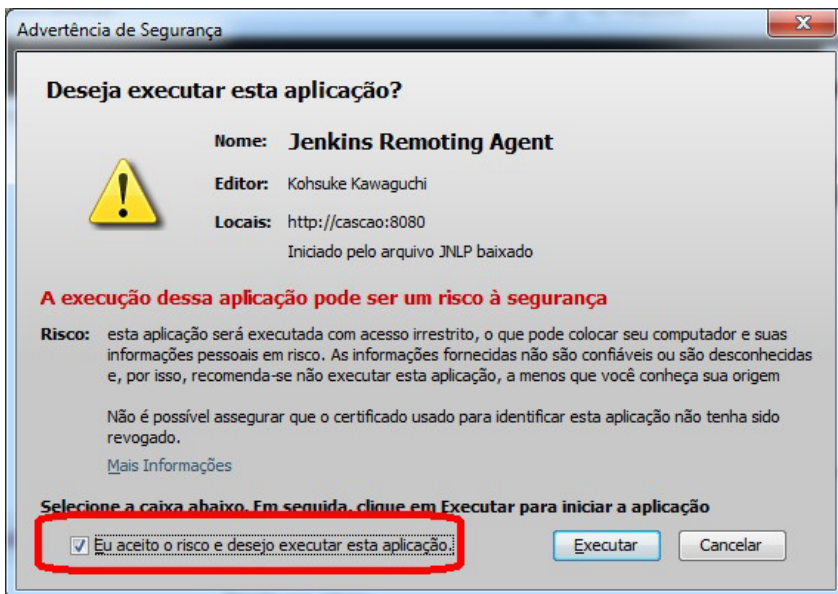


Figura 10.10: Registrar o slave no Jenkins – parte 2

Se tudo deu certo, aparecerá uma janela feita em Java Swing com a mensagem `Connected` e, na mesma tela em que clicamos no botão `Launch`, aparecerá apenas a mensagem `Conectado via JNLP agente`.

Se quisermos instalar o slave do Jenkins como um serviço do Windows, basta clicarmos na opção destacada na figura:

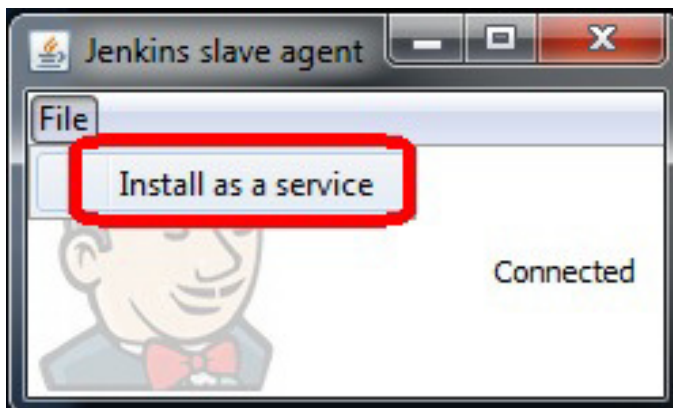


Figura 10.11: Registrar o slave no Jenkins como serviço

10.5 CONFIGURANDO TESTES NO INTERNET EXPLORER

Vamos criar um novo job `minhasmoedas-testes-it-no-IE` como cópia do `minhasmoedas-war`. Em seguida, alteraremos a parte do Maven e trocaremos o `clean install` (que faz os testes do Selenium com o Firefox) por `clean install -DwebBrowser=ie`, para fazer os testes com o Internet Explorer.

Nós já cadastramos e registramos o slave, e já criamos um job para testar nossa aplicação com o IE, agora só falta ligar tudo isso: configurar o nosso job para usar a máquina slave em vez da do servidor.

Isso é feito inicialmente instalando o plugin `Node and Label parameter plugin`, e depois ativando a opção `Este build é parametrizado` no job e, na opção `Adicionar parâmetro` -> `Node`, escolhendo o nó de desktop para rodar os builds desse job:

☒ Este build é parametrizado

Node

Name

Default nodes

master

desktop

Possible nodes

ALL (no restriction)

master

desktop

The nodes used when job gets executed

The nodes available for selection

☐ Run next build only if build is unstable ☐ Run next build only for successful builds, etc.

☐ Allow multi node selection ☒ Disallow multi node selection

In case of multi node selection, only for successful builds, etc.

Node eligibility

All Nodes

Figura 10.12: Escolhendo o nó desktop para rodar o job

Agora, com a máquina desktop ligada, o job pode ser disparado remotamente pelo Jenkins quando desejarmos.

Depois de uma execução de um build, o Internet Explorer abrirá no desktop e acessará as duas páginas do teste do Selenium. Também serão criadas screenshots dentro do workspace, como mostra a figura:

Workspace minhasmoedas-testes-it-no-IE em desktop



Figura 10.13: Screenshots do IE

10.6 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Instalar e configurar o plugins para o Selenium e builds em slaves funcionarem;
- Criar e configurar os testes no Selenium;
- Adicionar e configurar um nó (slave);
- Registrar uma máquina slave;
- Executar um build em uma máquina slave.

Depois que Bruno configurou os testes e o ambiente todos do Selenium, ele conseguiu criar os testes automáticos com o Firefox, e não precisa testar manualmente. Além disso, com o job rodando na máquina slave, ele não precisa sair de seu desktop Ubuntu Linux apenas para testar o site em um Internet Explorer, basta ele acompanhar o resultado do job. Se alguma nova tela aparecer no sistema, com certeza Bruno adicionará um teste de Selenium para ela, para não perder a consistência de seus testes.

No próximo capítulo, vamos aprender algumas dicas importantes e plugins imperdíveis para usar em nosso ambiente.

PLUGINS NINJA E DICAS

Dependendo do uso do Jenkins e do seu hardware, talvez seja necessário algum ajuste para melhorar a performance do seu ambiente.

Vamos partir do básico, pelo hardware da máquina, qual a melhor estratégia:

- **CPU** – é interessante usar um valor múltiplo do número de processadores para a configuração do Número de executores , por exemplo, uma máquina quadcore poderia receber o valor 8;
- **Memória** – para aumentar a memória *heap*, ajuste o parâmetro `JENKINS_JAVA_OPTIONS` em `/etc/sysconfig/jenkins` , por exemplo, adicionando o valor `-Xmx1024m` (no caso do Windows, as configurações ficam no arquivo `jenkins.xml`);
- **Hard disk** – depois de muitos builds, espaço pode ser um problema. Portanto, vamos ativar em cada build a opção `Descartar builds antigos` e colocar um valor baixo no `Máximo de builds para manter` .

Em seguida, vamos usar os plugins sugeridos para tirar melhor proveito do Jenkins.

11.1 PLUGINS USADOS NESTE LIVRO

Vamos listar os plugins usados neste livro como referência:

- **Xvfb plugin**

Utilizado para o funcionamento do Selenium em ambiente UNIX (<http://wiki.jenkins-ci.org/display/JENKINS/Xvfb+Plugin>).

- **Promoted Builds plugin**

Utilizado para promover os builds (<http://wiki.jenkins-ci.org/display/JENKINS/Promoted+Builds+Plugin>).

- **Deploy to container Plugin**

Utilizado para deploy em servidores de aplicação: Tomcat, JBoss e outros (<http://wiki.jenkins-ci.org/display/JENKINS/Deploy+Plugin>).

- **Delivery Pipeline Plugin**

Utilizado para visualizar os pipelines (<https://wiki.jenkins-ci.org/display/JENKINS/Delivery+Pipeline+Plugin>).

- **Artifactory Plugin**

Utilizado para se integrar ao Artifactory (<http://wiki.jenkins-ci.org/display/JENKINS/Artifactory+Plugin>).

- **Groovy Plugin**

Permite rodar as tasks do Groovy (<https://wiki.jenkins-ci.org/display/JENKINS/Groovy+Postbuild+Plugin>).

- **Gradle Plugin**

Permite rodar rotinas em Groovy (<https://wiki.jenkins-ci.org/display/JENKINS/Gradle+plugin>).

11.2 PLUGINS RECOMENDADOS

- **Backup plugin**

Faz backup e restaura os arquivos de configuração (<https://wiki.jenkins-ci.org/display/JENKINS/Backup+Plugin>).

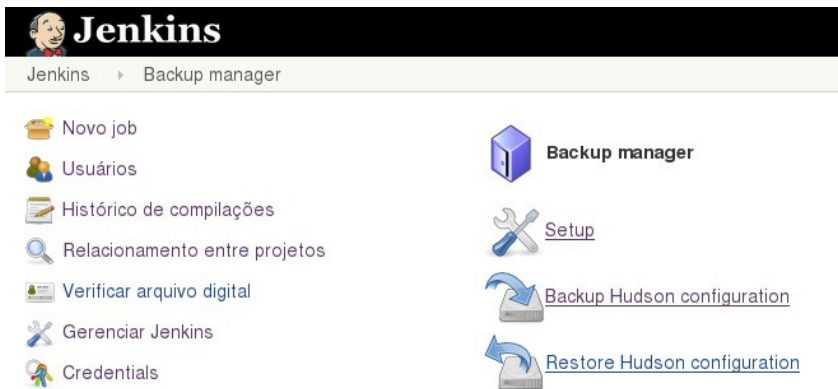


Figura 11.1: Opções do plugin de backup

- **Build Timeout Plugin**

Define um timeout para os builds (<https://wiki.jenkins-ci.org/display/JENKINS/Build-timeout+Plugin>).

Ambiente de build

☐ Start Xvfb before the build, and shut it down after.

☒ Abort the build if it's stuck

Time-out strategy

Absolute

Timeout minutes

10

Time-out variable

Set a build timeout environment variable

Time-out actions

Add action

Figura 11.2: Opções do plugin de timeout

• Disk-usage Plugin

Exibe o uso do espaço em disco (<https://wiki.jenkins-ci.org/display/JENKINS/Disk+Usage+Plugin>).

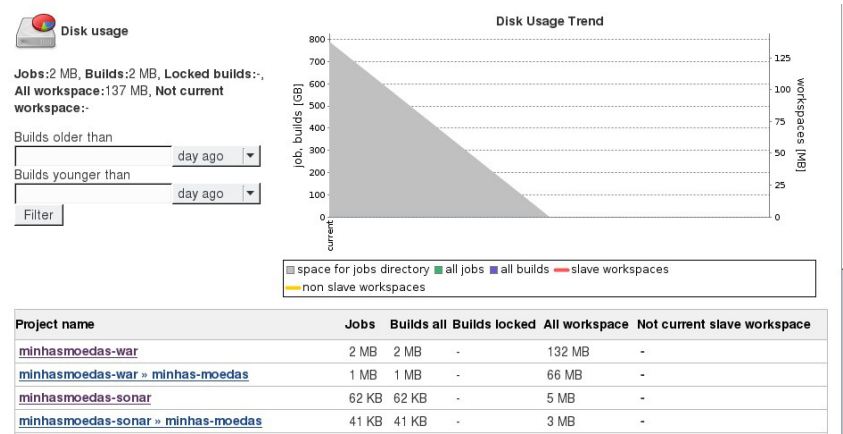


Figura 11.3: Opções do plugin de disk-usage

• Job Configuration History Plugin

Grava o histórico de suas configurações, permitindo comparar e restaurar alterações antigas.

Job Configuration History

minhasmoedas-artifactory

						Show Diffs	
Date ↑	Operation	User	Show File	Restore old config		File A	File B
2015-10-30_00-49-13	Changed	anonymous	 View as XML (RAW)			☐	☒
2015-10-30_00-49-12	Changed	anonymous	 View as XML (RAW)			☒	☐
2015-10-30_00-49-11	Changed	anonymous	 View as XML (RAW)			☐	☐
						Show Diffs	

Figura 11.4: Comparando configurações do job minhasmoedas-artifactory

11.3 PRÓXIMOS PASSOS

Certifique-se de que aprendeu como:

- Listar os principais plugins do Jenkins;
- Entender e aplicar as dicas para melhorar o ambiente.

Depois que Bruno instalou esses plugins, ele conseguiu muito mais do que o Jenkins pode oferecer, melhorando o seu trabalho e evitando problemas de lentidão.

No próximo capítulo, vamos aprender a criar o nosso próprio plugin do Jenkins.

CRIANDO O SEU PLUGIN

Apesar de existirem mais de mil opções, pode haver a necessidade de criarmos a nossa própria solução, ou seja, o nosso plugin personalizado.

Bruno precisava de um plugin para testar o tempo de conexão do banco de dados, mas como não encontrou nada parecido, resolveu implementar o seu próprio plugin.

Para entender como o desenvolvimento de plugins funciona, precisamos entender o Jelly, que é um framework que é um componente principal do Jenkins. É importante também entender que, apesar de o *runtime* ser recomendado, o JVM mais recente para desenvolvimento é recomendado usar a JVM versão 1.7.

12.1 JELLY

O Jelly é uma ferramenta da Apache que transforma um XML em um código executável. Ele é feito em uma engine script com Java e XML.

Esse framework será usado para mapear as telas para as rotinas criadas. O seu site oficial é <http://commons.apache.org/proper/commons-jelly/>.

12.2 CRIANDO UM PLUGIN SIMPLES

Vamos criar um plugin simples que faz um teste de uma conexão ao banco de dados e exibe o resultado no build dos jobs. O código-fonte está disponível em <https://github.com/boaglio/jenkins-plugin-database-connection-speed>.

12.3 CODIFICANDO O PLUGIN

O nosso plugin tem muita coisa pronta. Na verdade, ele aproveita toda a estrutura do Jenkins e apenas implementamos somente o mínimo necessário.

No projeto Java, especificamos no arquivo `pom.xml` que se trata de um plugin definindo, e que o pai desse projeto é o pacote de plugins do Jenkins. Com isso, toda a dependência de plugins será vinculada.

```
<parent>
<groupId>org.jenkins-ci.plugins</groupId>
<artifactId>plugin</artifactId>
<version>1.580.1</version>
</parent>
```

Vamos começar entendendo a estrutura da classe `TestConnection`, que testa a conexão, já que o que é importante não é seu conteúdo, e sim como os valores se integram à implementação do plugin.

```
public class TestConnection {
...
    public static void inicia
        (String url,
         String usuario,
         String senha,
         String dbType,
         boolean detalhes,
         BuildListener listener) throws Exception {

        /* testa a conexao */
        ...
    }
}
```

Exibimos o tempo gasto da conexão do banco de dados em milissegundos:

```
public static long diff() {  
    ...  
}
```

E imprimimos no console do build uma linha tracejada.

```
public static void finaliza() {  
    ...  
}
```

Quando usamos um plugin, percebemos que ele possui duas opções distintas de configuração: as particulares de cada job e as globais.

Na sua codificação, temos exatamente a mesma coisa. Por exemplo, a proposta do nosso plugin é bem simples, e ela tem apenas uma opção global de exibir mais detalhes, como mostra a figura a seguir.

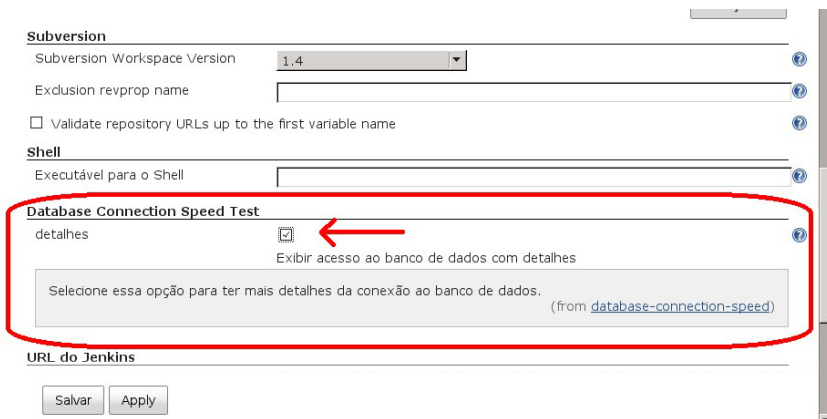


Figura 12.1: Opção de exibir mais detalhes da conexão

Em termos de HTML, trata-se de um simples checkbox. Essa tela é representada no Jelly da seguinte maneira:

```
<j:jelly>
```

```

<f:section title="Database Connection Speed Test">
  <f:entry title="detalhes" field="detalhes"
    description=
      "Exibir acesso ao banco de dados com detalhes">
    <f:checkbox />
  </f:entry>
</f:section>
</j:jelly>

```

Definimos um checkbox (`<f:checkbox />`) na tela que terá o seu valor atribuído à variável `detalhes` . Em seguida, temos a configuração dentro do job, que definimos informações da conexão do banco de dados, como mostra a figura adiante. Em termos de HTML, temos um combo e três campos de texto.

Figura 12.2: Configurações da base no job

Inicialmente, definimos o tipo de banco de dados na variável `dbType` .

```

<j:jelly>
  <f:entry name="dbType" title="Tipo de banco de dados"
    field="dbType">
    <select name="dbType">
      <option value="mysql">MySQL</option>
      <option value="oracle">Oracle</option>
      <option value="sqlserver">SQL Server</option>
    </select>
  </f:entry>

```

Então, definimos a URL de conexão JDBC para a variável `url` .

```

<f:entry title="url" field="url">

```

```
<f:textbox />
</f:entry>
```

Depois, o usuário do banco de dados para a variável `usuario` .

```
<f:entry title="usuario" field="usuario">
  <f:textbox />
</f:entry>
```

E, finalmente, a senha do usuário do banco de dados para a variável `senha` .

```
<f:entry title="senha" field="senha">
  <f:password />
</f:entry>
</j:jelly>
```

Com isso, a entrada de dados está definida. Agora, precisamos apenas trabalhar com os valores das variáveis e executar o comportamento esperado, que é o teste de conexão ao banco de dados. Isso também é feito de uma maneira bem simples, a nossa classe deve estender `hudson.tasks.Builder` .

```
public class ConnectionTestBuilder extends Builder {
  ...
```

Em seguida, criamos um construtor com os mesmos nomes e tipos das variáveis particulares de cada job.

```
@DataBoundConstructor
public ConnectionTestBuilder
  (String dbType,String url,String usuario,String senha) {
  this.dbType = dbType;
  this.url = url;
  this.usuario = usuario;
  this.senha = senha;
}
```

E essa classe deve ter uma `InnerClass` (classe dentro de uma classe), que contém as variáveis globais.

```
@Extension
public static final class DescriptorImpl
  extends BuildStepDescriptor<Builder> {
```

```
private boolean detalhes;

public boolean getDetalhes() {
    return detalhes;
}
```

E, finalmente, devemos implementar o método `perform`, que receberá nossas variáveis e outras do Jenkins, como a variável `listener`, que é utilizada para exibir valores no resultado do build dos jobs.

No nosso exemplo, chamaremos a classe `TestConnection` (que faz toda a "mágica" do plugin) passando todas as variáveis necessárias.

```
@Override
public boolean perform
(AbstractBuild build,
 Launcher launcher,
 BuildListener listener) {

    try {
        TestConnection.inicia
            (url, usuario, senha, dbType, getDescriptor()
                .getDetalhes(), listener);
    }
```

Em seguida, exibimos o tempo gasto na conexão de banco chamando o método `diff`:

```
listener.getLogger().println(" Tempo: "
    + TestConnection.diff() + " milisegundos ");
```

Finalmente, terminamos o *output* do plugin chamando o método `finaliza`, terminando o método `perform` e retornando `true`.

```
TestConnection.finaliza();
} catch (Exception e) {
    e.printStackTrace();
}
return true;
}
```

Isso encerra o desenvolvimento do plugin. Agora, é necessário gerar o pacote para usarmos na instalação.

O pacote compilado do nosso projeto é um JAR, mas o Jenkins trabalha com um formato diferente, que é o *hpi* (*hudson plug in*). Este contém o JAR do plugin junto com um arquivo XML, com informações de licença de uso. Esse arquivo tem a extensão *hpi*, mas na verdade tem o formato de um arquivo ZIP, e nós geramos com `mvn clean package install`.

12.4 INSTALANDO O PLUGIN

A instalação do plugin deve ser feita na aba **Avançado** do Gerenciador de plugins via upload do arquivo *hpi*, conforme mostra a imagem:

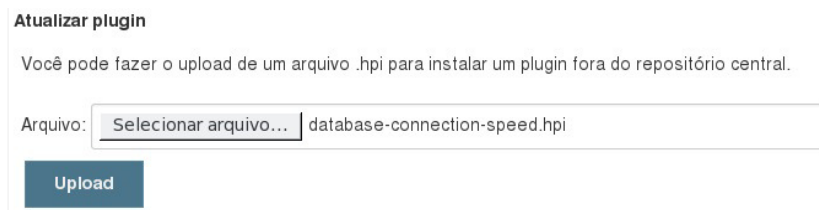


Figura 12.3: Instalação do plugin

Depois de instalar o plugin, precisamos reiniciar o Jenkins.

12.5 EXECUTANDO O PLUGIN

Depois de adicionar à configuração de um job a opção de **Build de Testar o acesso ao banco de dados**, ao executarmos o job, visualizamos no resultado de cada execução do build o output do teste de conexão, como mostra a figura a seguir:

```
At revision 20
no change for http://cascao/svn/boagliorep/minhasmoedas since the previous build
-----
Testando conexao [oracle] detalhada: minhasmoedas@jdbc:oracle:thin:@cascao:1521:o12c
Carregando driver...
Buscando a data de hoje...
Resultado do SQL: 2015-11-01
Finalizando...
Tempo: 1119 milisegundos
-----
[Gradle] - Launching build.
[workspace] $ /opt/gradle/latest/bin/gradle -x compileJava -x testClasses -x compileTe
-Pflyway.user=minhasmoedas_aceite -Pflyway.password=aceite
-Pflyway.url=jdbc:oracle:thin:@cascao:1521:012C -i clean processResources flywayMigrai
Starting Build
```

Figura 12.4: Resultado do plugin no build

O nosso plugin está pronto!

12.6 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- Conceito do Jelly;
- Como criar um plugin simples.

Depois que Bruno instalou o Jenkins em sua empresa, ele não achou nenhum plugin que atendesse às suas necessidades, portanto, ele teve que aprender um pouco de Jelly e também a criar um plugin customizado. Bruno não aprenderá mais nada neste livro, e sua história termina aqui. Mas, felizmente, existem outras excelentes fontes de informação do Jenkins.

No próximo capítulo, vamos aprender as fontes para continuar os estudos do Jenkins.

CONTINUE SEUS ESTUDOS

Agora em diante, para aprimorar os conhecimentos no Jenkins, participe dos grupos:

- Usuários do Jenkins — <https://groups.google.com/forum/#!forum/jenkinsci-br>
- Usuários deste livro de Jenkins — <http://forum.casadocodigo.com.br/>

Acompanhe os principais blogs:

- Blog oficial — <http://jenkins-ci.org/node>
- Informações das conferências de usuários de Jenkins pelo mundo — <http://jenkins-ci.org/category/tags/juc>

Acompanhe as novidades nas redes sociais:

- Twitter oficial Jenkins — <https://twitter.com/jenkinsci>
- Twitter oficial Jenkins Brasil — https://twitter.com/jenkins_br

E participe também dos encontros oficiais. Quase todo ano tem pelo menos um no Brasil. Saiba mais em <http://www.meetup.com/jenkinsmeetup/>.

Quem sabe nos encontros da comunidade Jenkins ou na net,

você conheça o Bruno. Sim, acredite, ele existe, mas não exatamente como foi descrito neste livro! ;)

Apêndices

APÊNDICE A – INSTALANDO O SVN

A instalação explicada a seguir foi feita em um Ubuntu 15.04. Talvez o seu ambiente seja um pouco diferente, mas os ajustes no roteiro serão mínimos.

Instalamos o servidor Subversion com os comandos:

```
> sudo apt-get update
> sudo apt-get install subversion apache2
   libapache2-svn apache2-utils
```

Em seguida, vamos criar um repositório chamado `boagliorep` com os comandos:

```
> sudo mkdir -p /svn/repos/
> sudo svnadmin create /svn/repos/boagliorep
> sudo chown -R www-data:www-data /svn/repos/boagliorep
```

Agora, adicionamos a configuração no Apache para acessar o repositório via HTTP, criando o arquivo `/etc/apache2/sites-available/boagliorep.conf`.

Dentro dele, adicionamos as diretivas do Apache para mapear o diretório `/svn`:

```
<Location /svn>
```

```
DAV svn
SVNParentPath /svn/repos/
AuthType Basic
AuthName "Repositorio"
AuthUserFile /etc/svnpasswd
Require valid-user
</Location>
```

Esse arquivo referencia um arquivo de senhas em `/etc/svnpasswd`, que conterà todos os usuários que podem acessar o SVN.

Vamos cadastrar um usuário chamado `usuario` com o comando adiante, informando a sua senha também:

```
> sudo htpasswd -cm /etc/svnpasswd usuario
New password:
Re-type new password:
Adding password for user usuario
>
> chown www-data:www-data /etc/svnpasswd
```

Ativaremos esse site criado com o comando:

```
> sudo a2ensite boagliorep
Enabling site boagliorep.
To activate the new configuration, you need to run:
    service apache2 reload
```

E, finalmente, vamos reiniciar o serviço do Apache com o comando:

```
sudo service apache2 reload
```

Para testarmos o novo ambiente, abrimos o browser na URL <http://localhost/svn/boagliorep>. Inicialmente, será exibida uma janela para autenticar, semelhante a essa:

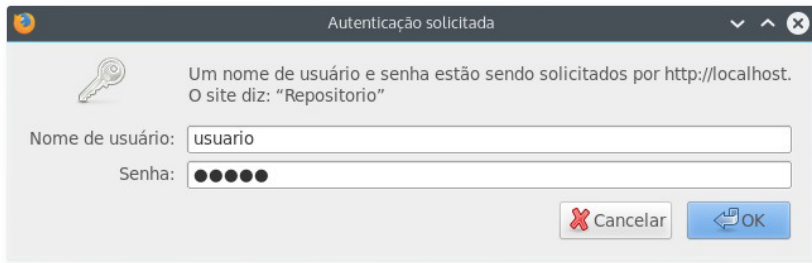


Figura 14.1: Autenticação via HTTP no SVN

Depois de autenticado, conseguimos visualizar o repositório vazio, sem nenhuma *revision*.

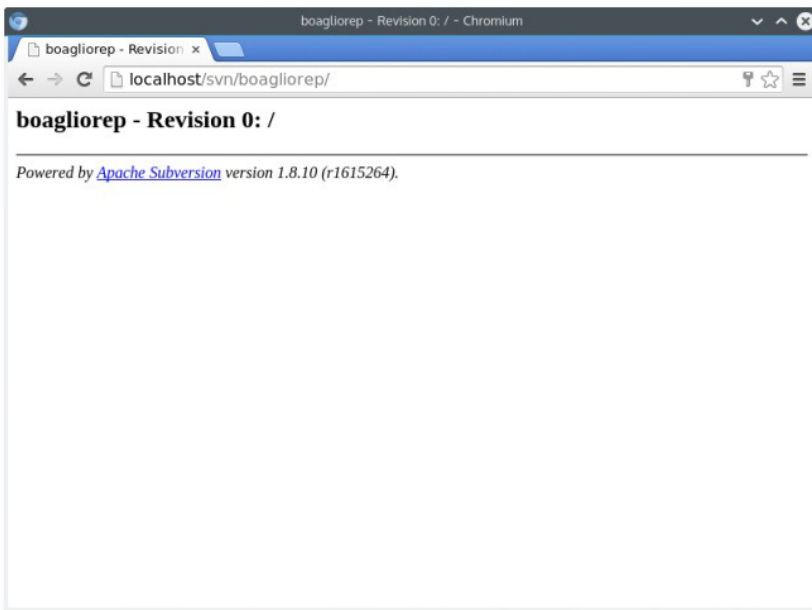


Figura 14.2: Repositório vazio

Indo além

Esse tutorial é uma adaptação desse <http://bit.do/svn1> (blog do Krizna). Para mais informações sobre o uso do SVN, acesse

<http://bit.do/svn2> (blog do Samuel Feitosa).

APÊNDICE B – INSTALANDO O ARTIFACTORY

A instalação explicada a seguir foi feita em um Ubuntu 15.0. Talvez o seu ambiente seja um pouco diferente, mas os ajustes no roteiro serão mínimos.

Não existe uma opção automática de instalação, portanto, é necessário baixar o arquivo no formato ZIP da última versão, em <http://www.jfrog.com/open-source/>.

```
> cd /opt  
> sudo unzip /tmp/artifactory-3.8.0.zip
```

A porta padrão do Artifactory é 8081. Se for necessário mudar esse valor, basta alterar o arquivo `/opt/artifactory-3.8.0/tomcat/conf/server.xml`. Instalaremos como serviço:

```
> cd /opt/artifactory-3.8.0/bin  
> sudo ./installService.sh  
Installing artifactory as a Unix service that will run as  
user artifactory  
Installing artifactory with home /opt/artifactory-3.8.0  
Creating user artifactory...creating... DONE  
  
Checking configuration link and files
```

```

in /etc/opt/jfrog/artifactory...
Moving configuration dir /opt/artifactory-3.8.0/etc
/opt/artifactory-3.8.0/etc.original...
creating the link and updating dir... DONE
** INFO: Please edit the files in /etc/opt/jfrog/artifactory
to set the correct environment
Especially /etc/opt/jfrog/artifactory/default that defines
ARTIFACTORY_HOME, JAVA_HOME and JAVA_OPTIONS
DONE
Setting file permissions... DONE

***** SUCCESS *****
Installation of Artifactory completed

Please check /etc/opt/jfrog/artifactory,
/opt/artifactory-3.8.0/tomcat
and /opt/artifactory-3.8.0 folders
Please check /etc/init.d/artifactory startup script

you can now check installation by running:
> service artifactory check (or /etc/init.d/artifactory check)

Then activate artifactory with:
> service artifactory start (or /etc/init.d/artifactory start)

```

O serviço foi instalado com sucesso. Então, vamos verificar os parâmetros do Artifactory com o comando:

```

> sudo service artifactory check
Checking arguments to Artifactory:
ARTIFACTORY_HOME = /opt/artifactory-3.8.0
ARTIFACTORY_USER = artifactory
TOMCAT_HOME      = /opt/artifactory-3.8.0/tomcat
ARTIFACTORY_PID  = /opt/artifactory-3.8.0/run/artifactory.pid
JAVA_HOME        =
JAVA_OPTIONS     = -server -Xms512m -Xmx2g -Xss256k

```

O valor `JAVA_HOME` aparece em branco. Para usar o JVM do Ubuntu, edite arquivo `/etc/opt/jfrog/artifactory/default` e adicione a linha:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/
```

E rodamos novamente o comando, agora o `JAVA_HOME` vem preenchido:

```
> sudo service artifactory check
Checking arguments to Artifactory:
ARTIFACTORY_HOME = /opt/artifactory-3.8.0
ARTIFACTORY_USER = artifactory
TOMCAT_HOME      = /opt/artifactory-3.8.0/tomcat
ARTIFACTORY_PID  = /opt/artifactory-3.8.0/run/artifactory.pid
JAVA_HOME        = /usr/lib/jvm/java-8-oracle
JAVA_OPTIONS     = -server -Xms512m -Xmx2g -Xss256k
```

Iniciamos o Artifactory com o comando:

```
> sudo service artifactory start
```

Para testarmos o novo ambiente, abrimos o browser na URL <http://localhost:8081/>.

A tela inicial será semelhante a essa:

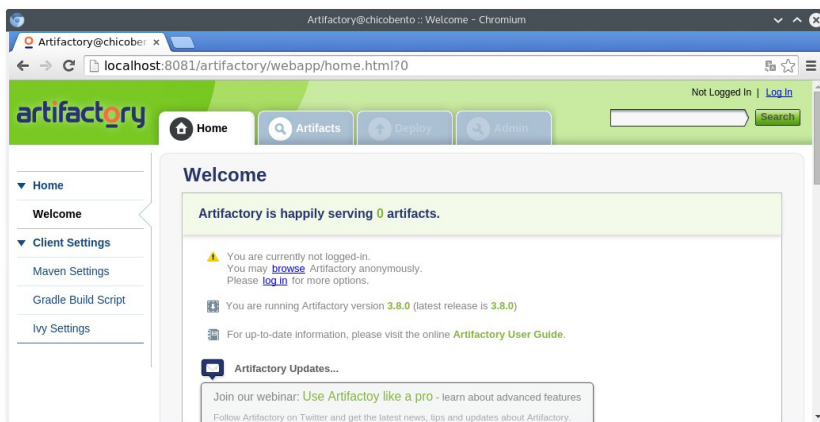


Figura 15.1: Tela inicial do Artifactory

Para entrar no sistema, informe o usuário `admin` e a senha `password` :

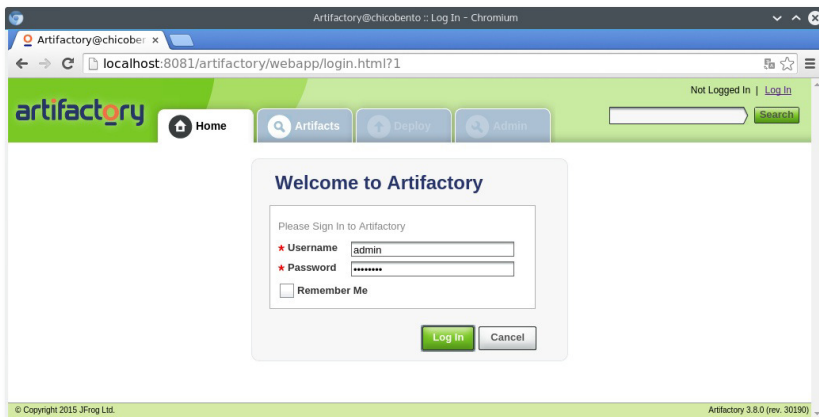


Figura 15.2: Autenticação no Artifactory

Indo além

Para mais informações do Artifactory, acesse o site oficial (<http://bit.do/artifactory>).

APÊNDICE C – MAVEN

16.1 O QUE É O MAVEN E COMO USÁ-LO?

O que é?

O Apache Maven, ou simplesmente Maven, é uma ferramenta de automação de compilação utilizada principalmente em projetos Java, semelhante ao Apache Ant, mas com conceitos diferentes. O seu site oficial é <https://maven.apache.org/>.

Como usá-lo?

No Maven, o projeto é definido por um arquivo que fica na raiz do seu projeto, chamado **POM** (*Project Object Model* – Modelo de Objeto de Projeto – `pom.xml`).

Veja um exemplo de um arquivo `pom.xml` bem simples que precisa da biblioteca do MySQL:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.meudominio</groupId>
  <artifactId>meuprojeto</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <!-- dependencia do MySQL -->
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.35</version>
    </dependency>
  </dependencies>
</project>
```

```
</dependencies>  
</project>
```

Dentro do POM, especificamos as dependências do projeto (bibliotecas de que ele precisa) e, opcionalmente, plugins para estender alguma funcionalidade.

Com um POM definido, conseguimos fazer uma série de tarefas predefinidas no Maven, como compilar o seu projeto, rodar testes, gerar um pacote JAR / WAR / EAR, gerar um site HTML estático com as informações do projeto, entre outras coisas.

Quase todas as bibliotecas que forem necessárias para o projeto estarão cadastradas no repositório oficial do Maven, catalogadas por versão e disponíveis para download. O Maven baixa desse repositório todas as bibliotecas necessárias para o funcionamento dele próprio e dos seus projetos.

As bibliotecas no repositório do Maven são chamadas de artefatos. Consulte os disponíveis em <http://search.maven.org>.

Sempre que for necessário, o Maven baixará um artefato do repositório oficial e, em seguida, copiará para um repositório local na sua máquina. Assim, da próxima vez que o Maven precisar desse mesmo artefato, ele buscará primeiro no repositório local e usará a internet somente se precisar buscar algo novo.

Existem algumas opções semelhantes ao Maven que têm alguns diferenciais, mas a maioria delas não possui um repositório de artefatos organizado e acaba utilizando o do Maven mesmo, já que ele não oferece nenhuma restrição. Se o seu projeto for open source, seguindo um procedimento simples, o seu pacote pode ser publicado no repositório do Maven sem problema nenhum.

Maven e Jenkins

Junto com Jenkins, o Maven auxilia no processo de build dos projetos, e os plugins existentes no Maven podem ser usados pelo Jenkins para tarefas específicas.

16.2 COMO INSTALÁ-LO?

A instalação do Maven no Ubuntu é feita com o comando `apt-get install maven`, conforme o exemplo a seguir:

```
> sudo apt-get install maven
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Os NOVOS pacotes a seguir serão instalados:
  maven
É preciso baixar 1.281 kB de arquivos.
Obter:1 http://br.archive.ubuntu.com/ubuntu/
  vivid/universe maven all 3.0.5-3 [1.281 kB]
Baixados 1.281 kB em 0s (2.465 kB/s)
A seleccionar pacote anteriormente não seleccionado maven.
A descompactar maven (3.0.5-3) ...
Configurando maven (3.0.5-3) ...
```

Para ver a versão instalada, use a flag `version` conforme o exemplo:

```
> mvn -version
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.8.0_45, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-oracle/jre
Default locale: pt_BR, platform encoding: UTF-8
OS name: "linux", version: "3.19.0-20-generic",
arch: "amd64", family: "unix"
```

16.3 FASES DO MAVEN

O ciclo de vida do Maven possuem as fases predefinidas, e elas podem ser executadas com o comando `mvn <fase>`. Os exemplos das fases mais comuns são:

- `clean` – limpa os arquivos temporários do projeto;
- `compile` – compila o projeto;
- `package` – faz o build do pacote definido no POM;
- `install` – faz o build e instala o pacote gerado no repositório local.

Veja um exemplo de execução de uma fase:

```
> mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building minhas-moedas 1.0.0
[INFO] -----
[INFO]
[INFO] Deleting /home/fb/workspace-jenkins/minhasmoedas/target
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.482 s
[INFO] Finished at: 2015-06-21T19:45:06-03:00
[INFO] Final Memory: 6M/109M
[INFO] -----
>
```

Indo além

O Maven possibilita fazer a documentação do seu sistema em HTML, dividir o seu projeto em módulos, como também controlar as versões de bibliotecas de seu sistema.

O site oficial possui uma excelente documentação. Consulte também o seu repositório online para pesquisar as versões de bibliotecas disponíveis (<http://mvnrepository.com/>).

APÊNDICE D – SONAR

17.1 O QUE É O SONAR E COMO USÁ-LO?

O que é?

O Sonar (ou SonarQube) é uma agregador de métricas que pode ser usado para medir a qualidade do código do seu sistema.

Ele é feito em Java e roda métricas para diversas linguagens (como Java, C++, JavaScript e CSS); a maioria delas de forma gratuita, e algumas pagas. O seu site oficial é <http://www.sonarqube.org/>.

Diversas métricas geradas em um projeto Java apontam falhas e sugerem melhorias:

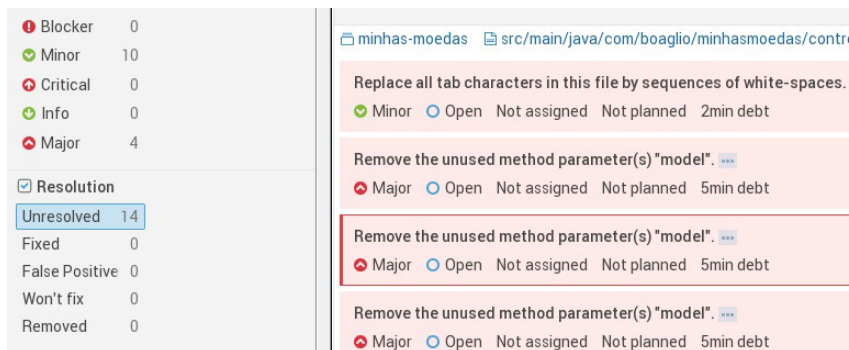


Figura 17.1: Problemas apontados pelo Sonar

Como usá-lo?

O uso do Sonar consiste em duas partes:

1. Gerar métricas por meio do plugin do Maven (`mvn sonar:sonar`);
2. Consultar as métricas geradas no site do Sonar.

Para gerar as métricas em projetos Java, usamos o plugin do Maven; para as linguagens restantes, é utilizado o SonarQube Runner.

Nos nossos projetos, precisamos definir no arquivo de configurações do Maven – (diretório-`HOME`)\.`m2`\`settings.xml` –, ou no Jenkins – `/var/lib/jenkins/.m2/settings.xml` –, as informações de acesso ao banco de dados do Sonar.

No exemplo a seguir, estamos considerando o nosso exemplo em que armazenamos as informações do Sonar dentro do Oracle:

```
<settings>
<profiles>
  <profile>
    <id>sonar</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <sonar.jdbc.url>
        jdbc:oracle:thin:@localhost:1521:o12c
      </sonar.jdbc.url>
      <sonar.jdbc.username>sonar</sonar.jdbc.username>
      <sonar.jdbc.password>sonar</sonar.jdbc.password>
      <sonar.host.url>http://cascao:9000</sonar.host.url>
    </properties>
  </profile>
</profiles>
</settings>
```

17.2 COMO INSTALAR?

Instalamos o servidor do Sonar inicialmente baixando do site e descompactando em um diretório local.

```
> cd /opt
> sudo unzip /tmp/sonarqube-5.1.2.zip
```

Em seguida, dentro do diretório `conf/sonar.properties`, configuramos o banco de dados que o Sonar gravará as métricas – no nosso caso, usaremos o Oracle.

```
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
sonar.jdbc.url=jdbc:oracle:thin:@localhost:1521:o12c
```

Assim, copiamos algum driver da Oracle para o diretório `extensions/jdbc-driver/oracle/`. Os drivers estão disponíveis no site da Oracle, ou no repositório do Maven (<http://bit.do/oraclejdbc6>).

17.3 INICIANDO O SONAR

Para subirmos o serviço do Sonar, basta acessarmos o diretório `/opt/sonarqube-5.1.2/bin/linux-x86-64` e iniciar o serviço com a opção `console`.

```
> cd /opt/sonarqube-5.1.2/bin/linux-x86-64
> ./sonar.sh console
Running SonarQube...
wrapper| --> Wrapper Started as Console
wrapper| Launching a JVM...
jvm 1 | Wrapper (Version 3.2.3)
                                     http://wrapper.tanukisoftware.org
jvm 1 | Copyright 1999-2006 Tanuki Software, Inc.
jvm 1 | All Rights Reserved.
jvm 1 | 2015.10.16 00:57:41 INFO app[o.s.p.m.JavaProcessLauncher]
jvm 1 | 2015.10.16 00:57:47 INFO app[o.s.p.m.Monitor]
jvm 1 | 2015.10.16 00:57:47 INFO app[o.s.p.m.JavaProcessLauncher]
jvm 1 | 2015.10.16 00:58:08 INFO app[o.s.p.m.Monitor]
                                     Process[web] is up
```

Em seguida, acessamos o Sonar no endereço

<http://localhost:9000>.

17.4 MELHORANDO O SONAR

O Sonar possui dezenas de plugins que melhoram as suas métricas e aumentam as funcionalidades existentes.

No nosso exemplo, vamos usar os dois mais comuns de projetos Java:

- SVN Plugin – verifica quem commitou as fontes (<http://docs.sonarqube.org/display/PLUG/SVN+Plugin>);
- Java Plugin – coloca mais de 300 regras para projetos Java (<http://docs.sonarqube.org/display/PLUG/Java+Plugin>)
-

Depois de baixarmos o JAR de cada um dos plugins, copiamos para o diretório `extensions/plugins/` e reiniciamos o Sonar.

Indo além

O Sonar oferece diversos plugins que oferecem mais métricas de análise de código e várias opções de relatórios. Consulte o site oficial para escolher o plugin desejado. A maioria deles é gratuita, mas alguns são pagos.

APÊNDICE E – SELENIUM

18.1 O QUE É O SELENIUM E COMO USÁ-LO?

O que é?

O Selenium é uma ferramenta open source de automação de web browsers. Isso significa que, com ele, conseguimos simular alguém navegando em um website. E como isso pode ser útil? Para testar os sistemas web!

O comportamento esperado de um site, como por exemplo, exibir uma mensagem de bem-vindo depois que o usuário se logar, pode ser uma excelente maneira de testar se o seu sistema está confiável depois que ele for publicado.

O seu site oficial é <http://www.seleniumhq.org/>.

Como usá-lo?

O Selenium pode ser usado por meio de uma ferramenta dele (Selenium IDE) ou de alguma linguagem de programação, como Java, PHP, Ruby ou Python.

Vamos ver um exemplo simples em Java, iniciando a chamada do driver do Firefox:

```
WebDriver driver = new FirefoxDriver();
```

Em seguida, usamos o driver para acessar a URL desejada:

```
driver.get("http://chicobento:18080/minhas-moedas/");
```

Depois, usando o método `getPageSource`, obtemos todo o HTML da página e testamos se ele possui o título da nossa aplicação:

```
if (driver.getPageSource().contains("Minhas Moedas")) {  
    System.out.println("homepage ok !");  
}  
driver.quit();
```

Indo além

O Selenium possui diversas utilidades interessantes, como usar também o Google Chrome ou o Internet Explorer, tirar screenshots das telas que navega, e também rodar seus testes em paralelo em várias instâncias em diversas máquinas. O site oficial tem vários exemplos e mais informações sobre as opções existentes dessa poderosa ferramenta.

APÊNDICE F – GRADLE E GROOVY

Para entender o que é o Gradle, precisamos conhecer o Groovy.

19.1 O QUE É O GROOVY?

O Groovy é uma linguagem de programação que funciona em cima da Java Virtual Machine, é orientada a scripts e proporciona uma sintaxe mais simples.

Veja um exemplo de um programa em Java:

```
public class Teste {  
    public static void main(String args[]) {  
        System.out.println("Teste");  
    }  
}
```

Agora, um exemplo do mesmo programa em Groovy:

```
println "Teste"
```

Em um exemplo mais interessante, temos uma verificação da hora atual, que exibe uma mensagem de OK apenas se for entre meia-noite e sete da manhã.

```
def now = new Date()  
def hora = now.getHours()  
  
println "Horario agora: "+hora  
if (hora>0 && hora< 7) {
```

```
println "Horario OK para deploy!"
}
else {
    println "Horario fora da janela, deploy cancelado!"
}
```

O Groovy tem uma ferramenta chamada Groovy Console para testar os scripts, conforme vemos na figura:

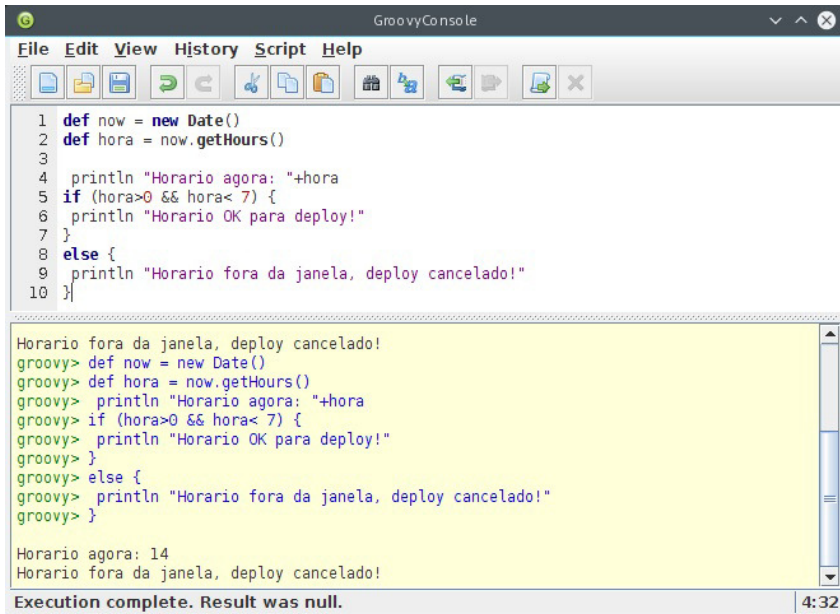


Figura 19.1: Groovy Console com script

Você pode encontrar mais informações sobre o Groovy em seu site oficial (<http://www.groovy-lang.org/>), ou no livro *Falando de Grails: Altíssima produtividade no desenvolvimento web*, de Henrique Lobo Weissmann (<http://www.casadocodigo.com.br/products/livro-grails>).

19.2 O QUE É O GRADLE?

O Gradle é uma ferramenta de automação de builds (semelhante

ao Maven) criada em Groovy. O seu site oficial é <http://gradle.org/>.

O arquivo de configuração padrão do Gradle é o `build.gradle`. Um projeto feito em Maven, com o seu arquivo `pom.xml` pode ser facilmente convertido para o Gradle:

```
> gradle init --type pom
:wrapper
:init
Maven to Gradle conversion is an incubating feature.

BUILD SUCCESSFUL

Total time: 3.401 secs
```

Esse comando criou o arquivo `build.gradle` para o seu projeto. Agora, ele está pronto para usar o Gradle.

Com ele, podemos rodar várias tasks, sendo uma delas o `build` (no exemplo com a opção de ignorar os testes):

```
> gradle build -x test
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:check
:build

BUILD SUCCESSFUL

Total time: 2.536 secs
```

Outro exemplo é chamar o `Flyway` (Apêndice G) para exibir informações das migrações executadas no banco de dados:

```
> gradle -x compileJava processResources
-x testClasses -x compileTestJava flywayInfo

:processResources UP-TO-DATE
:flywayInfo
+-----+-----+-----+-----+
| Version | Description          | Installed on          | State |
+-----+-----+-----+-----+
```

1	Cria tabela usuario	2015-07-15 23:07:01	Success	
2	Cadastra usuarios	2015-07-15 23:07:01	Success	
+-----+	+-----+	+-----+	+-----+	+-----+

BUILD SUCCESSFUL

Total time: 3.811 secs

Indo além

Para mais informações do Gradle com o Flyway, acesse o site <http://flywaydb.org/documentation/gradle/>.

APÊNDICE G – FLYWAY

20.1 O QUE É O FLYWAY E COMO USÁ-LO?

O que é?

O Flyway é uma ferramenta de migração de banco de dados. Além de ser open-source, é bem fácil de usar. O seu site oficial é <http://flywaydb.org>.

As suas migrações podem ser escritas em SQL ou em Java, e ele pode ser usado de várias maneiras, inclusive como um plugin do Apache Maven ou do Gradle.

20.2 COMO FUNCIONA?

O Flyway cria uma tabela chamada `schema_version` e, dentro dela, controla as alterações do seu banco de dados, cada uma delas tem uma versão diferente.

A alteração é executada via script, e o nome do arquivo tem uma ordem sequencial, com um padrão de nome que é a versão com a descrição, conforme a figura:

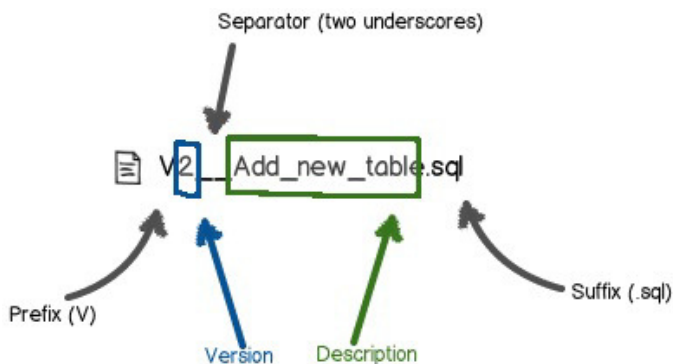


Figura 20.1: Padrão de nome do script do Flyway

Veja um exemplo:

```
V1__Cria_tabela_usuario.sql  
V2__Cadastra_usuarios.sql
```

Na tabela, ficam armazenados dessa maneira:

```
vers description  
-----  
1    Cria tabela usuario  
2    Cadastra usuarios
```

Dentro dos arquivos, existem os comandos SQL para criar os objetos do banco de dados ou fazer uma carga de dados.

O Flyway foi criado para trabalhar com *schema* e sem nenhum objeto, para começar do zero. Porém, ele pode trabalhar também com objetos já existentes.

Em um projeto Java com Maven, os scripts devem ser criados conforme a figura:

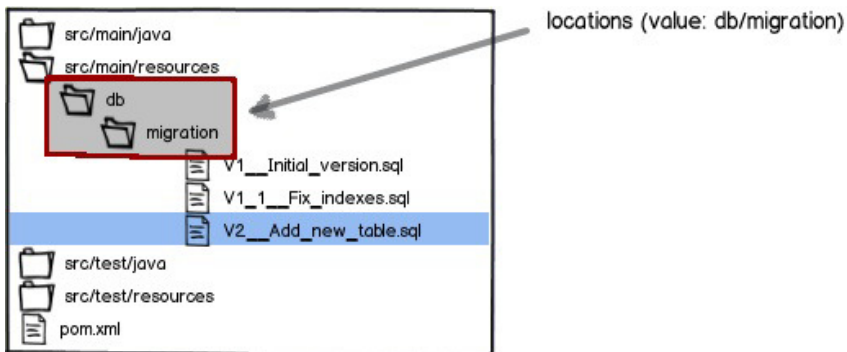


Figura 20.2: Diretório dos scripts do Flyway

Como usá-lo?

Os comandos existentes do Flyway são:

1. **Migrate** – faz a atualização do banco de dados;
2. **Clean** – limpa todo o seu banco de dados;
3. **Info** – informa quais alterações foram aplicadas em seu banco de dados;
4. **Validate** – valida as alterações aplicadas em seu banco de dados;
5. **Baseline** – é usada em banco de dados com objetos já existentes para definir um ponto de partida das alterações;
6. **Repair** – corrige a tabela de versões.

O Flyway pode ser usado em linha de comando, ou junto com o Gradle / Maven. Veja um exemplo:

```
flyway migrate
```

Indo além

O Flyway é bem simples e possui uma API para ser chamado internamente do seu projeto Java. Consulte a documentação no site oficial para mais informações.

APÊNDICE H – CENÁRIO GERAL

No ambiente dos servidores, são tantos serviços e endereços distintos que facilmente nos confundimos com alguns deles. Portanto, vamos resumir a seguir todos os tipos utilizados e suas respectivas funções.

21.1 MÁQUINAS DO AMBIENTE

- *cascao* - Apache Tomcat de desenvolvimento + Oracle Database + Artifactory + Sonar
- *chicobento* - Apache Tomcat de aceite e de produção

21.2 SERVIDORES DO SISTEMA

No Sistema Minhas Moedas, são:

- Desenvolvimento – <http://cascao:10000/>
- Aceite – <http://chicobento:18080/>
- Produção – <http://chicobento:28080/>

E no banco de dados, são:

- Desenvolvimento:

MINHASMOEDAS/MINHASMOEDAS@cascao:1521:012C

- Aceite:

MINHASMOEDAS/MINHASMOEDAS_ACEITE@cascao:1521:012C

- Produção:

MINHASMOEDAS_PROD/PROD@cascao:1521:012C

21.3 OUTROS SERVIDORES

- SVN – <http://cascao/svn/boagliorep/minhasmoedas/>
- Jenkins – <http://cascao:8080/>
- Artifactory – <http://cascao:8081/artifactory/>
- Sonar – <http://cascao:9000/>
- Banco de dados Oracle – host: cascao post: 1521
instance: o12c