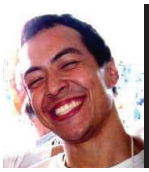
Aumentando a produtividade na administração de servidores Jenkins

Como fazer o essencial e o não tão essencial com o Jenkins. Aprenda a configurar ambientes distribuídos, criar scripts para automatizar tarefas e integrar o Jenkins com outras ferramentas como Puppet.

Jenkins é um servidor de integração contínua de código livre que é utilizado por empresas de todos os portes. Com ele, os desenvolvedores organizam builds, testes e até realizam o empacotamento e instalação das suas aplicações e, com o tempo, se torna uma ferramenta fundamental para os desenvolvedores. Mas conforme o número de jobs e builds vai aumentando, sua fila de builds começa a ficar cheia, e os desafios para administrá-lo também aumentam. Assim, neste artigo iremos apresentar como criar um ambiente distribuído para o Jenkins, automatizar diversas tarefas com Groovy e plug-ins, além de uma série de boas práticas e dicas.

Bruno P. Kinoshita | bruno@tupilabs.com
é engenheiro de software formado no Mackenzie, sócio de uma empresa de consultores independentes e um dos fundadores da TupiLabs – http://www.tupilabs.com. Dá palestras e escreve artigos no Brasil e no exterior, é committer da Apache e membro ativo de projetos de código livre. No tempo livre, pratica escalada, slack line, surfa, corre e desenha. Site pessoal http://www.kinoshita.eti.br e GitHub http://www.github.com/kinow.



Criar um job no Jenkins é simples e intuitivo. Você tem diversas opções de plug-ins e pode construir vários tipos de projetos diferentes (mobile, web, games, entre outros). Mas o que acontece quando você tem que criar 150 jobs de uma só vez? Ou quando você precisa criar servidores escravos (slaves) de forma automatizada? Ou ainda quando você precisa mudar a configuração de um plug-in apenas em um tipo de slave?

O Jenkins possui recursos como: Groovy; API externa em JSON e XML; API de plug-ins; Integração com outros sistemas; além de outros recursos que podem facilitar tarefas de manutenção, migrações e no gerenciamento do Jenkins e do seu ambiente de integração contínua.

Automatizando o Jenkins com Groovy

Groovy é uma linguagem dinâmica e simples de usar. Ela é baseada no Java (mas mais simples) e roda dentro da JVM. Assim, é possível executar trechos de código dentro da sua JVM e modificar objetos em tempo de execução, sem nem precisar compilar seus scripts.

O Jenkins utiliza Groovy de diversas maneiras. Quando escrevemos plug-ins, podemos usar Groovy tanto para o código quanto para a UI. A vantagem do Groovy é que podemos depurar a tela, diferente do Jelly.

Há também diversos plug-ins que nos permite executar o Groovy ou alguma DSL (domain specific language) com Groovy, como o Job DSL Plug-in. E há também o CLI do Jenkins, um cliente que nos permite executar diversos comandos no Jenkins, entre eles o groovysh, um shell para Groovy, e o groovy, e que também nos permite enviar um script para o Jenkins executar no servidor principal (master).

Para executar os exemplos deste artigo, você pode acessar http://<jenkins>/script ou criar um script e usar o CLI para executar o script diretamente no Jenkins. Tente fazer os dois para praticar.

Listando Plug-ins

Esse pode ser considerado o "hello world" de Groovy no Jenkins, porque ele é o exemplo que vem com o Jenkins. Se você for até a página de execução de scripts (http://<jenkins>/script) você encontra esse mesmo exemplo.

Listagem 1. Listando plug-ins com Groovy.

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

Limpando a fila de builds

Você pode encontrar vários outros scripts Groovy para o Jenkins na página de scripts do Scriptler Plug-in. Há um GoogleApp que guarda vários scripts que você pode utilizar no seu servidor. Esse exemplo foi desenvolvido pelo Kohsuke Kawaguchi, criador do Jenkins, e está disponível no GoogleApp.

Listagem 2. Limpando a fila de builds com Groovy.

```
import hudson.model.*;
Hudson.instance.computers.each { c ->
  c.executors.each { e ->
    e.interrupt();
  }
}
```

Jenkins em ambientes distribuídos

Quando você instala o Jenkins e começa a executar seus builds, por padrão ele roda apenas o master. Ou seja, um servidor apenas concentrando todos os builds. E esse servidor começa sempre com dois executores. Os executores são responsáveis por, como o nome diz, executar os builds.

Conforme o número de builds cresce, a fila do Jenkins cresce também e os executores não dão conta de executar todos os builds. Assim você acaba tendo que aumentar o número de executores. O problema é que não adianta você colocar 10 executores em uma máquina com duas cores (ou mesmo quatro

Groovy com Auto Complete no Eclipse
Com os plug-ins do Maven e do Groovy pra Eclipse, mais algumas configurações customizadas, podemos escrever scripts Groovy com auto complete no Eclipse. Para saber mais, acesse esse post www.tupilabs.com/pt-br/escrevendo-scripts-groovy-para-o-jenkins-usando-o-eclipse-com-auto-complete/ e veja o vídeo tutorial.

Scale out VS. Scale up

O Jenkins "scale out" melhor que "scale up". Isso quer dizer que no modelo de sistema distribuído do Jenkins, é melhor você criar vários slaves, mesmo em máquinas simples, do que você tentar criar um master em um supercomputador.

Neste vídeo o criador do Jenkins, Kohsuke Kawaguchi, fala mais sobre o modelo distribuído do Jenkins além de sobre outros tópicos de integração contínua e Jenkins, http://www.infoq.com/presentations/Whats-Next-in-Continuous-Integration

cores). A fila de builds continua com builds concorrendo pelos executores.

A solução é montar um ambiente distribuído com master e slaves. O master sendo responsável pela aplicação Web e por coordenar a execução de builds (bem como plug-ins e build steps) tanto no master como nos slaves.

Figura 1. *Página de gerenciamento do Jenkins.*

Criando um Slave simples

Há diversas formas de adicionar slaves no Jenkins. Você pode criar um slave simples (chamado também de dumb slave) que é controlado pelo master; criar um slave na nuvem com os plug-ins JClouds ou EC2; usar o Swarm Plug-in para auto-provisionamento de slaves ou até inventar um novo mecanismo específico para a configuração do seu projeto.

Vamos mostrar como criar um slave simples. Nosso master é um Linux e vamos criar um slave Windows que pode executar builds de aplicações nativas ou realizar testes com o IE.

O primeiro passo (figura 1) é acessar a página de configuração do Jenkins (http://<jenkins>/manage), e procurar pela opção para gerenciar slaves (Manage Nodes).

Na página de gerenciamento de slaves (figura 2) você vai encontrar o master sendo listado com suas configurações. Procure por um link na sua esquerda para adicionar novo slave (New Node).

Digite um nome para o novo slave e selecione Dumb Slave (figura 3).

Em seguida você será apresentado com um formulário para inserir mais parâmetros do slave (figura 4). Você ainda pode alterar o nome do slave e adicionar uma descrição. Mas os parâmetros mais importantes são número de executores, o file system remoto e os labels.

Por padrão, utilizamos dois executores, mas dependendo do número de cores disponíveis, este nú-

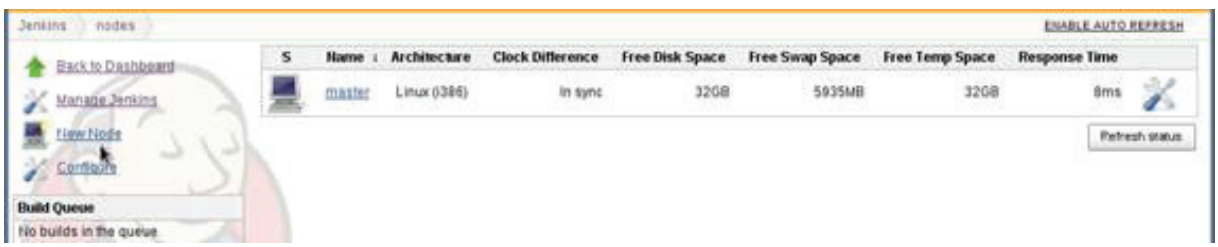


Figura 2. *Página de gerenciamento de slaves.*

Figura 3. *Criando um novo slave.*

mero pode ser aumentado. O file system remoto é o diretório onde o slave armazena arquivos serializados do master ou criados durante um build. Por fim, os labels são usados para a execução de builds, como veremos mais adiante.

Quando você salvar a listagem de nós (nodes) haverá um novo, além do master (figura 5). Porém o ícone do seu slave terá uma marca em vermelho, indicando que está offline. Você pode conferir o status dos slaves abaixo da fila de builds também.

Se você tiver selecionado para iniciar o slave pelo Java Web Start, basta executar javaws http://<jenkins>/computer/slave01/slave-agent.jnlp (ou baixar o arquivo JNLP e executá-lo da mesma forma). Se o slave for iniciado com sucesso, você verá uma janela como a da figura 6.

E quando voltarmos para a tela de listagem de

Figura 6. *Slave conectado no Windows.*

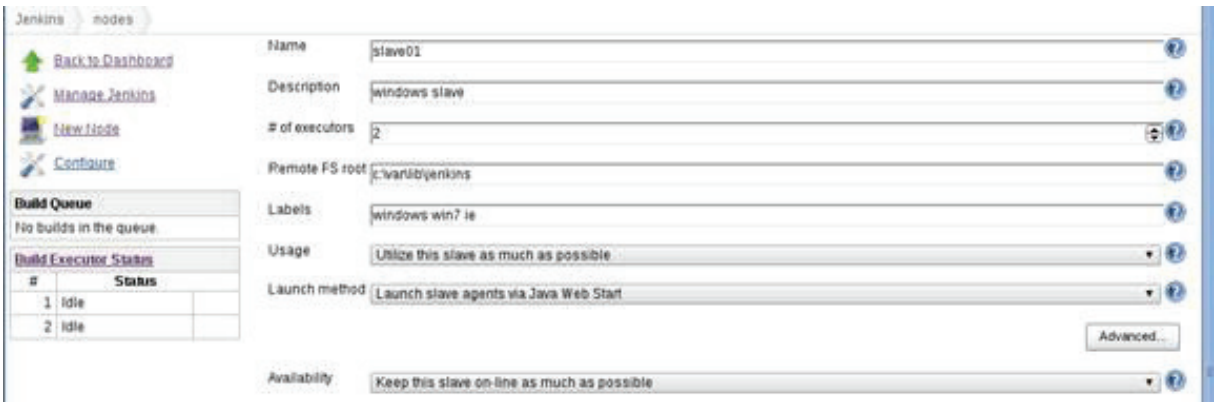


Figura 4. *Preenchendo parâmetros do slave.*

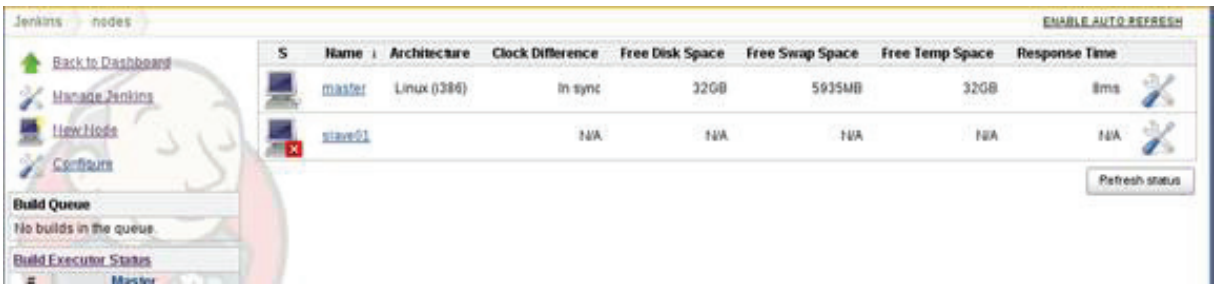


Figura 5. *Novo slave criado, mas não inicializado.*

Figura 7. *Slave conectado.*

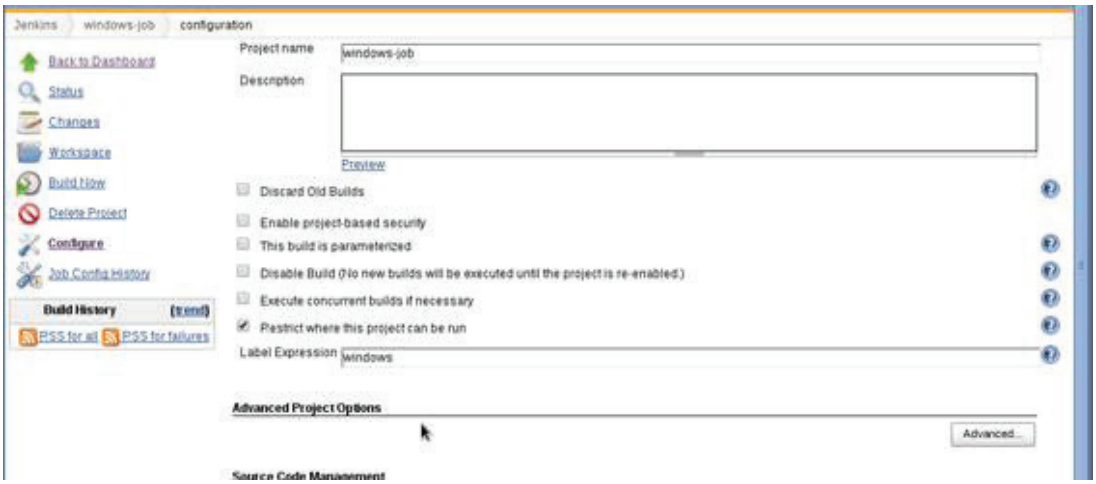


Figura 8. *Criando um job que utilize o slave.*

Figura 9. *Mostrando que o job foi executado remotamente.*

nós, veremos que o ícone do seu slave não tem mais a marca de offline e já possui as configurações do slave sendo listadas também (figura 7).

Por padrão o Jenkins direciona os builds para slaves. Porém jobs que foram criados antes de slaves são executados no master, para evitar erros de build caso o slave não esteja preparado para o build da aplicação.

Para testar seu novo slave, basta criar um novo job (figura 8). Ele já será executado no slave ou podemos também utilizar labels para restringir onde o job deve ser executado. É uma boa prática utilizar labels, e ainda pode ser combinada com jobs multi-configuração (aka Matrix Jobs).

Para saber onde o job executou (figura 9), podemos ver o console do Jenkins ou escrever um script Groovy que diga onde o job foi executado.

Outras maneiras de executar tarefas complexas

Há outros plug-ins para distribuir a execução dos seus jobs no Jenkins, como o Amazon EC2 Plug-in e o PBS Plug-in. Com o Amazon EC2 Plug-in, novos slaves são criados automaticamente em uma nuvem na Amazon quando o seu servidor fica sobrecarregado. Enquanto o PBS Plug-in nos permite executar e monitorar jobs de um cluster PBS (comum em workflows de bioinformática). Porém para a maioria dos casos o uso de slaves é suficiente para distribuir o trabalho no Jenkins.

Utilizando Jenkins com Puppet

Puppet é uma ferramenta de automatização que nos permite gerenciar a infraestrutura. Podemos utilizar diversos módulos existentes ou criar novos módulos para substituir tarefas repetitivas. Uma das vantagens de utilizar ferramentas como o Puppet (e.g.: Chef) é que podemos criar scripts simples, chamados manifestos, que contêm informações sobre seu ambiente. Como estes arquivos são texto simples, podemos versioná-los com Subversion, Git ou qualquer outro SCM da sua preferência.

O puppet tem um dashboard que mostra detalhes do seu ambiente, além do facter que reúne informações sobre a configuração da máquina e outros recursos úteis. O seu agente pode ser configurado para rodar periodicamente ou para ser executado somente sob demanda, num formato batch.

Além disso, podemos combinar o Jenkins e o Puppet de diferentes maneiras. Como criar uma configuração no Puppet para o master e slaves, incluindo versão do Jenkins e plug-ins, bem como JVM, Maven, Ant e outras ferramentas e serviços necessários. Assim é possível gerenciar instalações grandes com diversos tipos de slaves com diferentes requerimentos.

Podemos ainda utilizar o Puppet a partir do Jenkins, para levantar Virtual Machines e realizar deploy. Esse tipo de setup é comum quando se usa Continuous Delivery. Nas Listagens 3 e 4 temos exemplos de manifestos para um master e outro para os slaves do Jenkins.

Listagem 3. Manifesto Puppet para um master do Jenkins.

```
class jenkins {
  yumrepo {“jenkins”:
```

```
    baseurl  => "http://pkg.jenkins-ci.org/redhat",
    descr    => "Jenkins",
    enabled  => 1,
    gpgcheck => 1,
    gpgkey   => "http://pkg.jenkins-ci.org/redhat/jenkins-ci.
org.key",
  }
  package {"jenkins":
    ensure  => latest,
    require => Yumrepo["jenkins"]
  }
}
class git {
  yumrepo {"epel":
       baseurl    =>  "http://mirror.aarnet.edu.au/pub/epel/5/
i386",
    descr    => "Extra Packages for Enterprise Linux (EPEL)",
    enabled  => 1,
    gpgcheck => 1,
    gpgkey   => "http://keys.gnupg.net:11371/pks/lookup?se
arch=0x217521F6&op=get",
  }
  package {"git":
    ensure  => latest,
    require => Yumrepo["epel"]
  }
}
class java {
  package {"java-1.6.0-openjdk":
    ensure => latest
  }
}
include jenkins
include git
include java
```

Listagem 4. Manifesto para slaves do Jenkins.

```
class ci-ssh-slave {
    include packages::subversion
    include packages::vncserver # for Xvnc support
    group {
       "jenkins" :
          ensure  => present;
    }
    user {
       "jenkins" :
          gid     => "jenkins",
          ensure  => present,
          shell   => "/bin/bash",
          home    => "/home/jenkins",
          require => Group["jenkins"];
    }
    file {
       "/home/jenkins" :
          ensure      => directory,
          require     => User["jenkins"],
          owner       => "jenkins",
          group       => "jenkins";
       "/home/jenkins/.ssh" :
          ensure      => directory,
          require     => File["/home/jenkins"],
          owner       => "jenkins",
          group       => "jenkins";
    }
    ssh_authorized_key {
       "jenkins" :
          user    => "jenkins",
          ensure  => present,
          require => File["/home/jenkins/.ssh"],
          key     => "<KEY>",
          type    => "rsa",
          name    => "jenkins@company";
    }
}
```

Assim, podemos copiar o conteúdo dos manifestos em um arquivo e executar puppet apply <arquivo> -v para instalar Java, Jenkins e o Git. Este manifesto foi codificado para um servidor Red Hat. Normalmente realizamos algumas verificações no manifesto para ter certeza que ele funcionará com diferentes plataformas. O que esse manifesto faz é instalar o Git, o Java e o Jenkins. Para isso ele utiliza o repositório YUM do Jenkins, e há uma dependência entre o Jenkins e o Java (já que precisamos de Java para rodar o Jenkins).

O manifesto da Listagem 4 foi parcialmente copiado do repositório infra-puppet do Jenkins. Consulte nas referências bibliográficas para encontrar o link para mais scripts Puppet utilizados no projeto. Com isto, podemos adaptar vários desses scripts ao ambiente da sua empresa e automatizar a instalação do Jenkins e slaves.

Plug-ins essenciais (e os não tão essenciais também)

Há mais de 600 plug-ins no Jenkins hoje e, às vezes, podemos ficar tentado instalar diversos plug--ins. Porém essa é uma das causas mais frequentes de indisponibilidade do Jenkins. Versões mais novas dos plug-ins podem quebrar compatibilidade com versões antigas, causando erros ao carregar informações do disco.

O Jenkins não utiliza banco de dados para suas configurações. Ele guarda tudo em XMLs no disco usando o XStream. Quando instalamos plug-ins, alguns deles incluem dados nestes XMLs. Se o desenvolvedor do plug-in não tiver cuidado, é possível que ele quebre sua instância do Jenkins inserindo informações que o Jenkins não consegue carregar nos XMLs.

Nas páginas dos plug-ins no Wiki do Jenkins é

possível ver o número de downloads de cada plug-in. Prefira plug-ins mais antigos e com o número de downloads alto. Acompanhe também os bugs do plug-in para ver se não há nenhum grave. Com o tempo passaremos a ter uma lista de plug-ins favoritos e saber até a versão que funciona melhor com certa versão do Jenkins.

Abaixo há uma lista de plug-ins que podem ser úteis nos seus servidores Jenkins, independentemente de linguagem de programação ou do tipo do projeto.

1. Email-ext – Permite anexar logs ou utilizar um template para o corpo da mensagem quando notificar usuários de status de builds.
2. Build Timeout – Habilita um timeout (fixo ou elástico) nos builds. Assim não ficamos com builds prendendo sua fila por horas.
3. JobConfig History – Esse plug-in versiona a configuração dos seus jobs. Assim conseguimos auditar alterações na configuração dos seus jobs e até voltar versões.
4. Console Column – Adiciona um ícone para o último console na tabela de jobs. Muito útil quando precisamos abrir o console do último build do seu projeto rapidamente.
5. Extended Read Permission – Permite mostrar a configuração dos seus jobs mesmo sem estar logado, em modo de apenas leitura.

Mas há também plug-ins não tão essenciais, mas que são divertidos ou modificam de maneira simples o Jenkins.

1. Chuck Norris Plug-in - Insere o famoso Chuck Norris nos builds, parabenizando quando ocorrer um sucesso e se preparando para um roundhouse kick quando o build está quebrado.
2. Green Balls Plug-in – Mostra ícones verdes ao invés de ícones azuis para builds com sucesso.
3. Beer Plug-in – Quando precisamos realizar qualquer outra tarefa longe do computador, enquanto estamos monitorando nosso builds.
4. Continuous Integration Game - Plug-in que atribui pontos para os membros do time de desenvolvimento conforme eles vão evoluindo os seus builds.

Lista de plug-ins essenciais por Andrew Bayer

Andrew Bayer é um dos membros mais antigos do projeto. Trabalha na Cloudera, empresa que contribui com o Apache Hadoop, é um Apache Committer e está sempre no canal do #jenkins na Freenode com o nick @abayer. Ele fez uma ótima seleção de plug-ins que pode ser conferida em uma apresentação com o título "7 hábitos de usuários Jenkins altamente eficazes" http://www.slideshare.net/andrewbayer/7-habits-of-highly-effective-jenkins-users

Desenvolvendo Plug-ins

Por padrão, o Jenkins vem com alguns plug-ins embutidos, como o Maven Plug-in que fornece os templates de projetos para Maven, e o Subversion Plug-in que nos permite fazer check out de projetos de servidores Subversion no Jenkins.

Há mais de 600 plug-ins hoje para o Jenkins, e todo mês há plug-ins novos sendo criados e lançados no update site oficial do projeto. Porém há casos dentro de empresas que um build de certo projeto pode precisar de um plug-in também, mas que não faça sentido publicar este plug-in para a comunidade toda.

A maneira mais fácil de desenvolver plug-ins é seguindo o tutorial de plug-ins, do Wiki do Jenkins, e usando plug-ins existentes como base. Por exemplo, se quisermos que um plug-in rode um executável do Windows e gere algumas páginas customizadas no Jenkins, podemos usar o MSBuild Plug-in como exemplo.

Podemos escrever plug-ins em Java ou Ruby, porém nem todos os recursos do projeto estão disponíveis em Ruby ainda. Em Java, podemos utilizar tanto Maven quanto Gradle (maioria dos projetos utiliza Maven). E é possível utilizar Eclipse, Netbeans, IntelliJ ou o editor da sua preferência.

A parte Web do projeto utiliza o framework Stapler para MVC e REST, enquanto plug-ins utilizam Jelly, HTML e Groovy para escrever as telas que são apresentadas aos usuários.

Actions

Actions são objetos que respondem certas URLs. Quando criamos um plug-in é normal a necessidade de mostrarmos gráficos ou criar novas páginas. Normalmente utilizamos Actions que respondem a uma determinada URL, organizando objetos e usando Jelly ou Groovy para mostrar o conteúdo para o usuário.

A action abaixo é do Job DSL Plug-in. Quando o plug-in cria um novo Job usando uma DSL, ele adiciona essa action ao build. Essa Action possui um arquivo summary.groovy. Quando abrimos a página do build, o plug-in adiciona informações sobre os jobs criados ao sumário do build.

Listagem 5. Build Action do Job DSL Plug-in.

```
package javaposse.jobdsl.plugin;

import com.google.common.collect.Sets;
```

Vídeo-tutoriais sobre desenvolvimento de plug-ins para o Jenkins

Confira os vídeos tutoriais disponíveis no Youtube em português http://www.youtube.com/tupilabs. Eles seguem o mesmo script do tutorial de plug-ins do Jenkins, usando Java, Maven e Eclipse.

```
public Collection<GeneratedJob> getModifiedJobs() {
    return modifiedJobs;
  }

}
```

Listagem 6. Groovy da Build Action.

```
import hudson.model.Action;
import javaposse.jobdsl.dsl.GeneratedJob;
import java.util.Collection;
import java.util.Set;
 class GeneratedJobsBuildAction implements Action {
   public final Set<GeneratedJob> modifiedJobs;

   public GeneratedJobsBuildAction(
   Collection<GeneratedJob> modifiedJobs) {
      this.modifiedJobs = Sets.newHashSet(modifiedJobs);
   }

   /**
    * No task list item.
    */
   public String getIconFileName() {
      return null;
   }
   public String getDisplayName() {
      return "Generated Jobs";
   }
   public String getUrlName() {
      return "generatedJobs";
   }
```

```
package javaposse.jobdsl.plugin.
GeneratedJobsBuildAction;
import lib.LayoutTagLib
l=namespace(LayoutTagLib)
t=namespace("/lib/hudson")
st=namespace("jelly:stapler")
f=namespace("lib/form")
if (my?.modifiedJobs != null) {
   t.summary(icon:"folder.png") {
      raw("Generated Jobs:")
      ul(class:"jobList") {
         my.getModifiedJobs().each { af ->
            li() {
                  a(href:"${rootURL}/job/${af.jobName}/",
class:"model-link tl-tr") { raw(af.jobName) }
            }
         }
      }
   }
}
```

Figura 10. *Build Action em ação.*

Padrão Describable/Descriptor

No Jenkins quando escrevemos plug-ins, utilizamos sempre Extension Points. Existem diversos Extension Points, como o SCM para Source Code Management, utilizado pelos plug-ins Bazaar, Git e Subversion. Também é possível criar novos Extension Points para o seu plug-in. Uma atividade rotineira quando desenvolvemos plug-ins é a criação de formulários usados para preencher objetos Java. Há um padrão na API de plug-ins do Jenkins onde criamos um Descriptor que funciona como uma Factory para novos objetos (describables). O Descriptor é exposto por uma página de configurações (global ou de job).

Listagem 7. Extension Point SCM usado no Git Plug-in.

```
public class GitSCM extends SCM
implements Serializable {
   // ...
   @DataBoundConstructor

   public GitSCM(String scmName, List
   <UserRemoteConfig> userRemoteConfig, ...) {
      this.scmName = scmName;
      //...
   }
   // ...
}
```

Listagem 8. Descriptor do GitSCM.

```
@Extension
public static final class DescriptorImpl
extends SCMDescriptor<GitSCM> {


   private String gitExe;
   private String globalConfigName;
   private String globalConfigEmail;
   private boolean createAccountBasedOnEmail;

   public DescriptorImpl() {
      super(GitSCM.class, GitRepositoryBrowser.class);
      load();
   }

}
```

Na Listagem 7 a classe GitSCM estende SCM, um extension point para SCMs. E na Listagem 8 temos o Descriptor do GitSCM. No código-fonte do projeto, na pasta src/main/resources/hudson/plugin/git/GitSCM/ há dois arquivos importantes: *config.jelly* e *global.jelly*. O config.jelly cria um novo GitSCM, e o global.jelly cria um novo DescriptorImpl, com configurações globais que podem ser acessadas por instâncias do GitSCM.

Como obter ajuda e onde aprender mais

Participar do desenvolvimento do Jenkins é muito fácil, mesmo que você não programe Java. Há bindings para Ruby, bem como um trabalho que ainda não ficou pronto de bindings para Python. Também há testes Selenium e sempre são necessários testes antes de uma release LTS (Long Term Support). Mas se você programa em Java, você pode contribuir (ou se tornar o mantenedor) com plug-ins ou mesmo com o core.

No Wiki do Jenkins também encontramos muitas informações úteis, assim como no site da CloudBees. Os desenvolvedores e alguns usuários conversam no canal IRC #jenkins da FreeNode, mas há também mailing lists para usuários e desenvolvedores.

Recentemente foi criada uma lista para eventos e outra para segurança. Se você tiver um evento, ou se encontrar um bug crítico, consulte os links no final deste artigo para enviar mensagens para a lista correta. Há ainda uma lista de discussões em Português, criada após o evento Jenkins Meetup São Paulo em 2012. Consulte as referências bibliográficas no final deste artigo para o link.

Considerações finais

O Jenkins é uma ferramenta extremamente maleável, onde podemos inclusive re-empacotar e gerar sua própria distribuição, com outra cara e um conjunto de plug-ins embutidos por padrão. Como suas configurações ficam em XMLs no disco, é bem fácil realizar backups também.

E com um conjunto de computadores é possível montar um ambiente distribuído que suporte diversos builds em paralelo. Há diversos plug-ins para as mais variadas atividades, desde envio de notificações para celulares ou Skype, até execução de passos de construção (ou *build steps*) em clusters de alta disponibilidade.

Com o conjunto correto de plug-ins e com a adoção de boas práticas, é possível montar um ambiente resiliente e independente de linguagem de programação para compilar e testar seus projetos. Este artigo focou na administração de servidores Jenkins, mas você pode se interessar por outros artigos sobre *Continuous Integration, Continuous Delivery, Continuous Deployment, Builds Pipeline* e implantação (deploy) de aplicativos com o Jenkins.

/referências

> *Jenkins CI – http://www.jenkins-ci.org*

> *Jenkins: The Definitive Guide – John Fergusson Smart*

> *@jenkins-br Conta do Twitter brasileiro do Jenkins*

> *Lista de e-mails em português do Jenkins – https://groups.google.com/forum/?fromgroups#!forum/jenkinsci-br*

> *Administering Jenkins – https://wiki.jenkins-ci.org/display/JENKINS/Administering+Jenkins*

AUTOMATIZANDO O JENKINS COM GROOVY

> *Jenkins CLI – https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+CLI*

> *Groovy Beginners Tutorial – http://groovy.codehaus.org/Beginners+Tutorial*

> *Jenkins Groovy Plug-in – https://wiki.jenkins-ci.org/display/JENKINS/Groovy+Plugin*

> *Jenkins Scriptler Plug-in – https://wiki.jenkins-ci.org/display/JENKINS/Scriptler+Plugin*

> *Scriptler Google App – http://scriptlerweb.appspot.com/*

JENKINS EM AMBIENTES DISTRIBUÍDOS

> *Distributed builds – https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds*

> *Jenkins Amazon EC2 Plug-in – https://wiki.jenkins-ci.org/display/JENKINS/Amazon+EC2+Plugin*

> *BioUno PBS Plug-in – https://github.com/tupilabs/pbs-plugin*

> *Jenkins Swarm Plug-in – https://wiki.jenkins-ci.org/display/JENKINS/Swarm+Plugin*

UTILIZANDO JENKINS COM PUPPET

> *PuppetLabs Puppet – http://puppetlabs.com/*

> *Puppet no Jenkins com Ubuntu – https://wiki.jenkins-ci.org/display/JENKINS/Puppet*

> *Módulo puppet-jenkins – https://github.com/rtyler/puppet-jenkins*

> *Post do rtyler sobre Jenkins e Puppet – http://unethicalblogger.com/2011/12/28/jenkins-with-puppet.html*

> *Repositório de manifestos e módulos Puppet do Jenkins - https://github.com/jenkinsci/infra-puppet*

PLUG-INS ESSENCIAIS (E OS NÃO TÃO ESSENCIAIS TAMBÉM)

> *Jenkins E-mail-ext Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin*

> *Jenkins Build Timeout Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Build-timeout+Plugin*

> *Jenkins JobConfigHistory Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/JobConfigHistory+Plugin*

> *Jenkins Column View Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Console+Column+Plugin*

> *Jenkins Extended Read Permission Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Extended+Read+Permission+Plugin*

> *Jenkins ChuckNorris Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/ChuckNorris+Plugin*

> *Jenkins Green Balls Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Green+Balls*

> *Jenkins Beer Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Beer+Plugin*

> *Jenkins Continuous Integration Game Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/The+Continuous+Integration+Game+plugin*

DESENVOLVENDO PLUG-INS

> *Tutorial de desenvolvimento de Plug-ins - https://wiki.jenkins-ci.org/display/JENKINS/Plugin+tutorial*

> *Definindo um novo extension point - https://wiki.jenkins-ci.org/display/JENKINS/Defining+a+new+extension+point*

> *Post no blog da CloudBees sobre Actions - http://blog.cloudbees.com/2011/08/jenkins-internal-action-and-its.html*

PADRÃO DESCRIBABLE/DESCRIPTOR

> *Extension Points - https://wiki.jenkins-ci.org/display/JENKINS/Extension+points*

> *Jenkins Git Plug-in - https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin*

COMO OBTER AJUDA E ONDE APRENDER MAIS

> *Listas de e-mail do Jenkins - http://jenkins-ci.org/content/mailing-lists*

> *Plug-ins do Jenkins - https://wiki.jenkins-ci.org/display/JENKINS/Plugins*

> *FreeNode – http://www.freenode.net*