Cenários de Continuous Integration utilizando Cloud Computing

Continuous Integration using Cloud Computing

Joana Coelho Vigário
Dep. de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro
Aveiro, Portugal
joana.coelho@ua.pt

Cláudio Teixeira, Joaquim Sousa Pinto
Dep. de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro
Aveiro, Portugal
{claudio,jsp}@ua.pt

Resumo — **Hoje em dia os sistemas podem evoluir rapidamente, e a esse crescimento está associado, por exemplo, a adição de funcionalidades, ou mesmo a mudança de perspetiva do sistema, por requisito dos *stakeholders*. Consequentemente estas provocam um aumento do número de testes de s*oftware* desenvolvidos. Executar sequencialmente uma grande bateria de testes é impensável porque a execução pode levar horas. Contudo, os testes podem executar mais rapidamente num ambiente distribuído e com rápida disponibilização de sistemas previamente configurados, como é o caso de *Cloud Computing*. Cada vez mais se procura automação sobre todo o processo, incluindo compilação e execução de testes.**
Este artigo pretende demonstrar alguns cenários sobre a aplicação da prática *Continuous Integration* (CI) em Sistemas de Informação, com testes de s*oftware* executados em *Cloud Computing*. Assim, pretende-se explorar ao máximo as capacidades que a prática CI dá, para a automatização da compilação e execução de testes num Sistema de Informação.

Palavras Chave – Integração Contínua; arquitetura; Cloud Computing; compilação; testes; software; deployment.

Abstract — **Nowadays systems can evolve quickly, and to this growth is associated, for example, the addiction of new features, or even the change of system perspective, required by the *stakeholders*. Consequently, these cause an increase in the number of developed tests. Run a large battery of tests sequentially can take hours. However, tests can run faster in a distributed environment with rapid availability of pre-configured systems, such as Cloud Computing.**
This paper pretends to demonstrate some scenarios on the implementation of the practice Continuous Integration (CI) in Information Systems, with software tests performed on Cloud Computing. The main goal is explore the most of capacities that CI practice gives, for automating the build and testing in a Information System.

Keywords – Continuous Integration; architecture; Cloud Computing; build; testing; software; deployment.

I. INTRODUÇÃO

A. Enquadramento

À medida que um Sistema de Informação cresce, é necessário garantir que todas as componentes que o formam se comportam como esperado. O Sistema de Informação da Justiça de Cabo-Verde (SIJ) é o sistema base deste estudo, e é um projeto que está em produção há mais de um ano, e que neste momento já conta com uma bateria de mais de 3000 testes.

Os testes de *software* têm como principal objetivo a verificação do correto funcionamento de uma pequena parte de uma aplicação e assegurar o seu correto funcionamento quando futuras alterações sejam realizadas na mesma [1]. O sistema de controlo de versões de código (SCV) também é extremamente importante para que a equipa consiga trabalhar sobre a última versão do código, possa ver o histórico do mesmo e comparar versões de ficheiros. O SCV permite que vários elementos da equipa trabalhem em simultâneo sobre o repositório de dados, e por este facto é possível ocorrerem erros de integração de ficheiros (quando duas ou mais pessoas trabalham sobre o mesmo ficheiro).
Num universo de mais de 3000 testes é crucial ter uma arquitetura que ajude a preparação e execução dos testes no mínimo tempo possível. Devido à complexidade e vasta gama de testes do SIJ, antigamente a execução sequencial dos mesmos ocorria entre 10 a 14 horas [1]. A justificação para estes tempos elevados está relacionada com as condições do ambiente de execução dos testes: execução sequencial, o que era incomportável. Depois de um trabalho de análise e alteração de configuração de estrutura de testes, foi possível diminuir o tempo de execução para 2.5 a 3 horas [1], utilizando uma solução distribuída.

B. Motivação

Atualmente, o SIJ utiliza o Team Foundation Server (TFS) [2] para a gestão do ciclo de vida do *software*, a metodologia de desenvolvimento *Scrum* [3] e tem uma configuração de testes dinâmica, utilizando um controlador e agentes de testes do Visual Studio *(VS)* [4]. A melhoria do tempo de execução dos testes mencionada anteriormente deve-se a que a execução dos mesmos ocorre num ambiente distribuído de *Cloud Computing*, em máquinas virtuais (VMs). Para a criação e gestão das mesmas é utilizada a *framework "*OpenNebula*"* [5]. As VMs utilizadas na estrutura atual têm um número fixo, sendo sempre as mesmas, e o ambiente delas foi preparado com a configuração necessária no momento após a criação das

mesmas, manualmente. Estas VMs estão sempre ligadas na *Cloud*, consumindo muitos recursos da mesma, mesmo quando os testes não estão em execução.

C. Problema

Com o avançar do desenvolvimento de um sistema, várias funcionalidades são acrescentadas, ou pode ocorrer uma mudança da perspetiva do sistema, devido a alterações de requisitos por parte dos *stakeholders*. É necessário que o sistema tenha o funcionamento esperado com a integração das diversas componentes, fazendo a sua validação com testes. É esperado que quando uma nova funcionalidade é adicionada, não danifique o código anteriormente correto e funcional, e que todo o processo seja dinâmico. Para o *tester* seria útil que a execução dos testes fosse um processo automático, em todas as fases que o completam, desde a obtenção da última versão SCV, a compilação sem erros, a cópia dos ficheiros de interface/serviços/base de dados, a inicialização dos testes e a recolha dos resultados. Todas estas etapas automatizadas facilitam o trabalho da equipa de testes. A poupança dos recursos do OpenNebula também poderia ser conseguida pela automatização do controlo das VMs, isto é, apenas estariam a consumir os recursos nas horas da execução dos testes. Além da poupança dos recursos, a automatização do processo poderia ainda permitir que as VMs fossem criadas com a configuração necessária para a execução de testes, consoante o desejo do *tester*, permitindo uma elasticidade da *Cloud*. Isto é, o *tester* teria a liberdade de escolha do número de VMs que executariam os testes, fazendo assim a gestão da criação de VMs e controlando a rapidez de execução dos mesmos. Com esta estrutura, a rentabilidade do OpenNebula seria muito maior, principalmente durante a noite, sendo este horário mais favorável para a execução dos testes, pelo menor número de utilizadores.

II. ESTADO DE ARTE

A. Contextualização

Como mencionado anteriormente, o que se pretende obter é uma solução dinâmica e automatizada para execução de testes de um Sistema de Informação. Em 2000, Martin Fowler escreve sobre o tema *Continuous Integration* (CI) e explica as técnicas e uso desta prática. *Continuous Integration* é uma prática de desenvolvimento de *software* onde membros de uma equipa integram o seu trabalho frequentemente durante o dia, pelo menos uma vez. Cada uma destas integrações é verificada por uma compilação automática e pode ser validada por testes de integração. O processo de CI é apresentado na Figura 1.

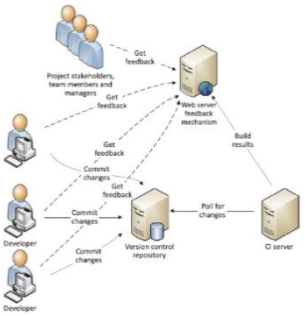

Figura 1 – Processo de CI [6]

Existem algumas estratégias de validação no mecanismo de CI para a compilação do projeto, como diária, contínua, semanalmente, entre outras [7]. Deste modo pode reduzir-se significativamente problemas de integração encontrados frequentemente pela operação de *merge* [8]. Com a realização das operações de *merge, update* e *check-in*/*commit* todos os dias, a integração poderá ser mais pequena e menos complexa, e será mais fácil de corrigir os conflitos. Uma das estratégias possíveis em CI é, por exemplo, garantir que o repositório é compilado antes e depois de cada *check-in*. Assim, caso após o *commit* de código, no projeto em questão, já não compilar, o programador deve tomar a correção do erro como prioridade. Tipicamente, além da simples compilação de código podem também ser executados testes unitários para despistagem de eventuais problemas. O servidor de CI interage com o repositório, pesquisando por mudanças, e se existirem, o servidor compila e testa o código. Os resultados são enviados para o sistema de feedback onde os membros da equipa podem ver os resultados.

Martin Fowler enumera as boas práticas que devem ser seguidas pela equipa para a prática de CI:

- Manter um único repositório de código;
- Automatizar a compilação;
- Fazer a nossa compilação *self-testing*;
- Todos os programadores fazem *commit* (ou *check-in*) para a versão main todos os dias;
- Cada *commit* (ou *check-in*) deve compilar a versão *main* na máquina de integração;
- Resolver compilações falhadas imediatamente;
- Manter a compilação rápida;
- Testar o projeto num ambiente que simule o ambiente de produção;
- Tornar mais fácil para qualquer um obter a última versão executável;
- Automatizar o *deployment* do projeto.

Martin considera que o melhor benefício que a prática CI dá é a redução do risco. Isto porque com sua utilização, os programadores já não investem tanto tempo na integração, já sabem em que ponto está o sistema, e sabem no momento se as suas contribuições para o sistema causam ou não erros, e se sim, então corrigem-nos antes de causar maiores erros no sistema completo. No livro *Continuous Integration: Improving Software Quality and Reducing Risk* é explorado mais pormenorizadamente o valor de CI. Da mesma opinião de

Martin, os autores do livro afirmam que o valor de CI é a redução de risco, a redução de processos manuais realizados repetidamente, a produção de *software* estável pronto em qualquer momento, uma melhor visibilidade do projeto e o estabelecimento de mais confiança no produto. Ao fazer várias integrações ao dia reduz-se os riscos como a descoberta tardia de erros, a falta de *software* coeso, de baixa qualidade e visibilidade do projeto. CI gera *software* pronto a publicar em qualquer momento, sendo este um dos aspetos mais relevantes da adoção desta prática. Por fim, no livro é mencionado que CI estabelece maior confiança no produto, porque a cada compilação, a equipa de desenvolvimento conhece o impacto das mudanças realizadas no código, e obtém os resultados dos testes executados, verificando o comportamento do *software*, para a obtenção de um produto testado funcionalmente [9]. De modo a conseguir adotar esta prática para a solução pretendida, é necessário conhecer ferramentas para depois ser possível escolher a que melhor se adequa ao problema. É essencial um SCV, servidor CI, gestor de compilação e de testes. Pode ainda ser completado por um mecanismo de feedback, uma ferramenta de análise de código, etc. Tendo em consideração o caso de estudo concreto, deve ser também avaliado se as ferramentas atualmente em utilização são as mais indicadas para a inclusão do processo de CI.

B. Análise de Ferramentas

1. Gestão de controlo de versões

Um SCV permite ver o histórico, fazer *update* ou *reverse*, de qualquer momento no tempo e de qualquer ficheiro no repositório, permitindo a comunicação entre os membros da equipa. Guarda ainda esquemas de base de dados, *scripts* e ficheiros de configuração de serviços e testes.

1.1. Team Foundation Server (TFS)

O TFS fornece um núcleo de funcionalidades para as equipas de desenvolvimento, como gestão de projeto, acompanhamento de itens de trabalho, gestão de casos de testes, automação de compilação, controlo de versão, relatórios, gestão de laboratório e ambiente de gestão de feedback [10]. É um servidor desenhado para qualquer interveniente do desenvolvimento, como programador, *tester*, arquiteto, gestor do projeto, etc.

1.2. Subversion + TortoiseSVN

O Subversion [11] é um sistema de controlo centralizado (à semelhança do TFS). Os programadores trabalham a partir do seu ambiente e necessitam de acesso ao servidor apenas na altura de fazer *commit* para o servidor. Esta ferramenta consegue detetar a informação de versões de ficheiros, diretórios e metadata, isto é, consegue saber o histórico entre mudanças e renomeação de diretórios, ao contrário de outros sistemas. Tem um *commit* atómico, ou seja, ou todo o conteúdo é submetido ou então as mudanças não são enviadas para o servidor [11].O TortoiseSVN é um cliente do Subversion para Windows que integra diretamente no Windows Explorer e permite as normais opções de um repositório.

1.3. Git + TortoiseGit

A ferramenta Git [12] é distribuída, ou seja, pode fazer-se *commit* para a versão local, sem haver a necessidade de conetividade ao servidor. O projeto tem o seu próprio repositório (privado) e um repositório comum para os elementos da equipa (público). O programador pode decidir quando integrar o seu código no repositório público. O Git permite que todos os programadores façam as operações habituais ou que haja um responsável que o faça. Em Windows, os utilizadores podem usar o instalador msysgit ou Cygwin e podem ainda usar como cliente o TortoiseGit. O TortoiseGit serve para o controlo de versão Git, implementado como uma interface de comandos Windows [13].

1.4. Resumo

Os SCVs apresentados não se diferenciam muito entre eles no que diz respeito às funcionalidades principais que a generalidade destas ferramentas oferece. Nessa perspetiva o TFS é o que mais se ajusta às qualidades pretendidas para a arquitetura em estudo, pelo simples facto de ser a ferramenta já em utilização e com a qual a equipa de desenvolvimento está mais familiarizada.

2. Servidor de Continuous Integration

Um servidor de CI guia o processo de CI, verificando mudanças no repositório e coordenando os passos do processo. Tipicamente tem o seu próprio mecanismo de feedback. Normalmente estes servidores são conduzidos por um ficheiro de configuração que especifica os passos a ter durante o processo de compilação e integração. É aconselhável ter uma máquina que apenas corra o servidor CI porque um processo corretamente criado deve ter o menor número de dependências possíveis [14].

2.1. Team Foundation Server (TFS)

O TFS é um sistema de gestão do ciclo de vida da aplicação. Orquestra todos os aspetos como o desenvolvimento do *software*, gestão de requisitos e projeto, desenvolvimento de testes e garantia de qualidade [15].

2.2. CruiseControl.NET

CruiseControl.NET (CCNet) é um servidor de CI, implementado usando .NET, que pode executar com consola ou serviço Windows. Esta versão é baseada numa versão da ferramenta em Java. O servidor automatiza o processo de integração por monitorizar diretamente o repositório de código, validando as mudanças de atualizações do código [16]. O CCNet pode gerir remotamente o processo usando um sistema chamado CCTray, que consegue aceder ao servidor remotamente para inicializar compilações ou apenas ser notificado da ocorrência de uma compilação.

2.3. Jenkins/Hudson

É um servidor de CI baseado em Java, usado para projetos desde .NET, Ruby, PHP, Grails, inclusive Java [17]. É uma tecnologia *open source*, tem diversos plugins para suporte à comunicação, execução de testes e integração com outros sistemas, e tem uma configuração simples baseada em interface web. O Jenkins executa a compilação do projeto via uma funcionalidade chamada *job*, que consiste na execução de uma série de tarefa [18].

2.4. Resumo

Os servidores apresentados registam algumas diferenças entre eles no que diz respeito à sua estrutura. No entanto, estas diferenças são mínimas se se considerar as funcionalidades

que se pretende usar. Assim, dada a decisão anterior no caso de estudo, relativamente ao TFS, optou-se por mantê-lo como servidor de CI.

3. Gestão de compilação

Para realizar a compilação do código é possível escolher, entre outras opções, o MSBuild e Nant. Permitem a utilização de *scripts* para compilar o projeto completo usando apenas um comando. Há sistemas que usam o MSBuild para a compilação do código e que usam o Nant para integrar outras ferramentas, como por exemplo para a execução dos testes unitários.

3.1. MSBuild

MSBuild é parte integrante da *framework* .NET e é o que mais se assemelha ao clique na opção "*compilar*" do VS. A sua configuração é definida por um documento XML. Não dá para todas as plataformas (limitado a .NET), tem algumas funcionalidades integradas e é ativamente desenvolvido [19]. O documento XML tem a definição de propriedades e os *targets* (*debug*, *release*, entre outros), onde cada *target* contém um número de tarefas [20].

3.2. Nant

Nant é desenhado a partir de uma ferramenta em Java chamada de Apache Ant. É uma ferramenta *open source* mantida por uma comunidade, que pode ser usada em qualquer plataforma e não é desenvolvido ativamente. É, à semelhança do MSBuild, controlado por um documento de configuração em XML. É uma boa opção para programadores que dominam Java e que são familiares com Ant [19].

3.3. Resumo

O MSBuild é mais adequando num ambiente .NET. Esta solução pode no entanto ser complementada com o Nant para execução de testes unitários em outros ambientes, ou na realização de tarefas específicas em que a sintaxe do Nant melhor se adeque.

4. Utilização do TFS para testes de software

O TFS possui mecanismos eficazes para a automação de testes. Um dos exemplos é a ferramenta MSTest. Esta ferramenta permite executar testes automatizados, ver os resultados dos testes e guardar os mesmos tanto no disco como no TFS. No sentido da automação, e por se pretender a execução distribuída dos testes, é possível recorrer às entidades "controlador" e "agente", associados ao TFS.

O controlador de testes (*Test Controller*) é o serviço responsável por controlar a execução de testes, isto é, publica os resultados dos mesmos e coordena a sessão destes para os agentes de testes. O agente de testes (*Test Agent*) é o serviço que está sempre ligado ao controlador, está instalado numa máquina de testes e que permite a sua execução e recolha de dados [21].

5. Utilização de Cloud Computing para CI

No livro *Cloud Computing Bible* [22], de Barrie Sosinky, o autor afirma que a "nuvem" já é utilizada há muito. O termo *Cloud Computing* refere-se a aplicações e serviços que são executados numa rede distribuída, usando recursos virtualizados e acedidos por padrões e protocolos da Internet.

Na estrutura do SIJ é utilizado a plataforma OpenNebula para a execução de testes de *software*. Esta plataforma oferece uma solução simples mas flexível e rica em funcionalidades para a gestão dos recursos associados às *clouds* [23]. O OpenNebula foi desenvolvido para que a integração de qualquer componente e/ou qualquer infraestrutura fosse simples. Assim, o resultado é um sistema modular que pode implementar uma variedade de arquiteturas e pode interagir com múltiplos serviços. É considerada uma Infraestrutura como Serviço (IaaS), dado que proporciona gestão de imagens e contexto, armazenamento, rede, monitorização e calendarização, virtualização, gestão de utilizadores e papéis [24].

Grande parte das funcionalidades do OpenNebula está disponível mediante serviços e API's, catalogadas internamente como *Cloud interfaces* e *System interfaces*. A primeira tem como propósito desenvolver ferramentas para o utilizador final, com um elevado nível de abstração das funcionalidades da *Cloud.* Esta interface, ainda que mais simples e concetualmente mais próxima das necessidades para este caso de estudo, foi descontinuada. A segunda expõe todas as funcionalidades do OpenNebula e é usada para adaptar e ajustar o comportamento do OpenNebula na infraestrutura desejada [25]. As *System interfaces* proporcionam API's de baixo nível. Estas API's de integração são a XML-RPC e a OpenNebula Cloud API (OCA). O OpenNebula oferece ainda *drivers* de interface, que permitem a interação entre o OpenNebula e a restante infraestrutura. As áreas cobertas pelos drivers são armazenamento, virtualização, monitorização, autorização e *networking*.

5.1. XML-RPC API

A interface XML-RPC é a interface primária do OpenNebula e expõe todas as funcionalidades da mesma. Esta interface é usada, por exemplo, para desenvolver bibliotecas especializadas para aplicações *Cloud* ou para os casos em que o programador necessita de uma interface de baixo nível para o núcleo do OpenNebula. Permite controlar e gerir qualquer recurso da *Cloud*, incluindo VMs, usando ações, como por exemplo desligar, retomar, reiniciar e arrancar VMs.

5.2. Ruby/Java OpenNebula Cloud API

A OCA está disponível em bibliotecas Ruby/Java, e permite uma integração simples, nestas linguagens, com o núcleo do OpenNebula. É composta por um conjunto de bibliotecas que facilitam a comunicação com a interface XML-RPC [26]. Esta API deve ser utilizada se se está a desenvolver uma ferramenta IaaS que necessita total acesso às funcionalidades do OpenNebula [25].

5.3. OneFlow API

A API OneFlow é um serviço RESTful para gerir, criar, controlar e monitorizar aplicações com várias camadas ou serviços compostos para VMs interligadas. Todos os dados são enviados/recebidos por JSON [25].

5.4. Resumo

As API's XML-RPC e OCA assentam na mesma estrutura, diferenciando-se na linguagem de comunicação, enquanto a OneFlow se baseia em pedidos HTTP por JSON. Deste modo, a API escolhida é a XML-RPC visto ser simples de utilização e providenciar as funcionalidades desejadas.

III. PROPOSTA DE SOLUÇÃO

A utilização da prática de CI e de *Cloud Computing* para automação e execução distribuída dos testes é essencial para a otimização do desenvolvimento de um sistema. Os testes permitem garantir que o sistema funciona como o esperado quando há adição de uma nova funcionalidade. Isto é, quando uma nova funcionalidade é adicionada, o funcionamento do sistema deve manter-se correto. Já a utilização de *Cloud Computing,* além da melhoria notória do tempo de execução dos testes (desde que executados em paralelo), permite a elasticidade do sistema. Isto quer dizer que o OpenNebula pode adaptar-se dinamicamente às necessidades impostas pelo *tester*, e assim conseguir adicionar e/ou remover VMs que executem os testes com mais rapidez, e ainda calendarizar as execuções do mesmo para obter resultados quando desejado. Todas as componentes abordadas neste artigo permitem ter conhecimento para o desenvolvimento da arquitetura pretendida. Assim, é necessário uma arquitetura com um editor de desenvolvimento, um SCV, um servidor de CI, uma ferramenta de compilação, de *deploy*, de testes e de *Cloud Computing*, como na Figura 1 – Processo de CI . A equipa realiza *commit*/*update* do código do SCV, em que há um servidor de CI que procura por mudanças no repositório. Se há mudanças, então o servidor puxa-as e faz a compilação do código. Além de compilar, faz a automação dos testes de *software* nas VMs.

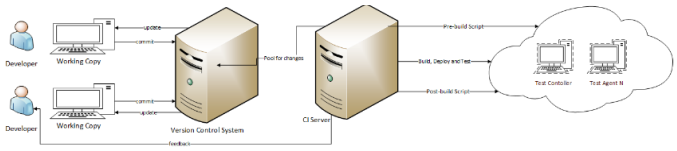

Figura 3 – Desenho da arquitetura de CI

Depois de definidas as componentes necessárias, é apresentada uma proposta de CI, tendo em consideração os objetivos pretendidos para a aplicação SIJ. Visto que o sistema é assente na plataforma .NET, foi selecionado para o editor de desenvolvimento o VS2013, com o SCV e servidor CI o TFS. O TFS é ótimo para o trabalho em equipa, os membros da equipa conseguem sempre estar a par das tarefas que têm de desempenhar. A compilação, o *deploy* e a execução de testes são executados pelas ferramentas MSBuild, MSDeploy e MSTest, respetivamente. Para a distribuição dos testes pelas VMs são utilizados o controlador e agentes de testes do VS. Além de compilados os ficheiros da aplicação Web, podem ainda ser compilados ficheiros de serviços e base de dados, dados estes fundamentais para o deploy do sistema.

Como parte essencial da nova proposta, pretende-se uma integração entre o servidor de CI e o OpenNebula, conseguindo deste modo uma gestão dinâmica e elástica das VMs. Esta integração é possível pela utilização de uma biblioteca .NET [27] que encapsula o acesso à API XML-RPC do OpenNebula. Foram analisadas várias possibilidades e a biblioteca de Charles Cook é a que melhor se adequa para a utilização de funcionalidades *core* do OpenNebula, assumindo que esta integração parte de um sistema/serviço em .NET. Esta biblioteca permite a realização de pedidos ao OpenNebula para gestão das VMs. Assim, propõe-se a criação de um serviço que consuma a biblioteca e que seja invocado na hora

```
[XmlRpcMethod("one.vm.action")]
Array oneVirtualMachineAction(string sessionSHA, string action, int
vmId);
```

Figura 2 - Método que comunica com API do OpenNebula

da validação de código. Atendendo ao fluxo de compilação do MSBuild, esse serviço deverá ser usado nos eventos de pós compilação, como ilustrado na Figura 3 - Chamada do serviço no evento de pós compilação.

```
<ItemDefinitionGroup>
   <PostBuildEvent>
      <Command>net start C:\Myfile\MyService.exe</Command>
      <Message>Stoping service MyService.exe</Message>
   </PostBuildEvent>
</ItemDefinitionGroup>
```

Figura 4 - Chamada do serviço no evento de pós compilação

Na Figura 4 - Método que comunica com API do OpenNebula é demonstrado o método que poderá comunicar com a API do OpenNebula para suspender a VM.

O serviço invoca funções para cada ação que se deseje realizar no OpenNebula. No seguinte exemplo (Figura 5 - Função para suspender uma máquina virtual) é possível ver a função invocada para suspender uma VM.

```
public static Array suspendVirtualMachine(string action, int vmId)
{
   //Exemplo para suspender a máquina virtual
   string rootMainUserHash = "userName:passWord";
   XmlRpcVirtualMachineManagement xrum = XmlRpcProxyGen.Create<
XmlRpcVirtualMachineManagement>();
   Array openNebulaReturnArr =
xrum.oneVirtualMachineAction(rootMainUserHash, action, vmId);
   return openNebulaReturnArr;
}
```

Figura 5 - Função para suspender uma máquina virtual

Por fim, é possível ver a arquitetura idealizada na Figura 6, que demonstra uma possível e melhor solução para automatização de processo de desenvolvimento e testes no SIJ.

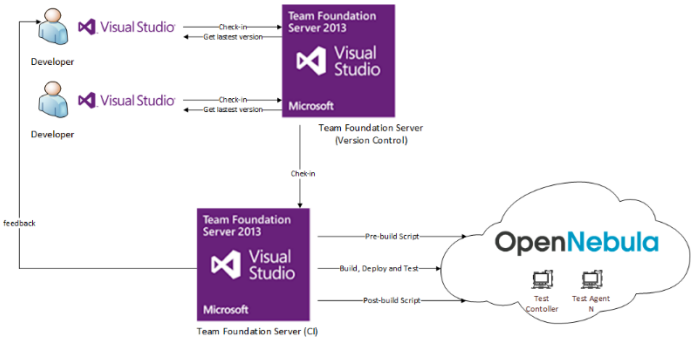

Figura 6 – Proposta de arquitetura de CI, para o SIJ

Com base nesta arquitetura, e tendo em consideração a utilização da API de gestão de VMs, será possível a ligação direta entre uma tarefa de validação do código (iniciada

manualmente ou agendada, de acordo com uma dada frequência de *commit,* uma determinada frequência temporal, etc.) e a existência de VMs em quantidade suficiente para a execução do teste, no menor espaço de tempo possível. Assim que os testes terminem, estas VMs serão novamente descartadas, libertando assim os recursos da estrutura de *cloud* privada, baseada no OpenNebula. A proposta de fluxo automatizado em cenários de *Cloud Computing* é apresentada na Figura 7. As alterações, relativamente ao modelo tradicional de CI, estão assinaladas na figura, com os retângulos preenchidos. Como se pode observar na mesma, a alteração proposta é mínima e diz respeito unicamente ao processo de instanciação e destruição das VMs em que o código será instalado e testado.

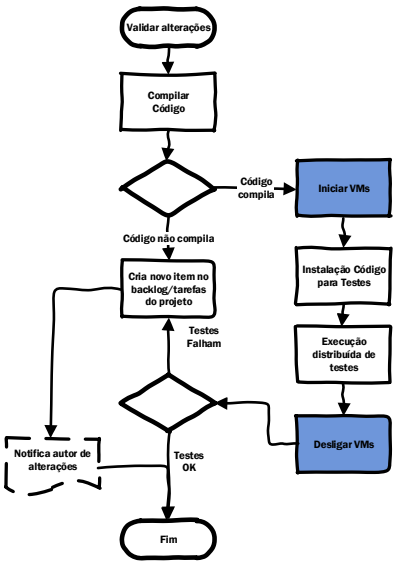

Figura 7 – Proposta de alteração de fluxo de CI,

IV. CONCLUSÕES

Um Sistema de Informação pode tornar-se muito complexo à medida que é desenvolvido e usado, e quando o seu processo de desenvolvimento e testes é um processo automatizado, os programadores ficam com mais tempo para desenvolver/corrigir funcionalidades do sistema. A utilização de *Continuous Integration* permite precisamente automatizar este processo. A arquitetura de desenvolvimento que o SIJ hoje apresenta não é uma solução de CI. Assim, propõe-se uma nova arquitetura para a automatização do processo de desenvolvimento do mesmo. A estrutura apresentada foi idealizada também para ser uma arquitetura modelar, isto é, facilmente se adapta a uma outra linguagem ou ferramenta, tanto de compilação como testes. A automação da compilação e *deploy* dos testes irá permitir que a execução corra automaticamente, preferencialmente num horário noturno, nas máquinas virtuais de teste. Pretende-se ter um pleno controlo das mesmas e que sejam alocados todos os recursos possíveis da *Cloud* apenas durante a execução dos testes, libertando-os no fim da sua execução, tirando partido do conceito de computação elástica associada a *Cloud Computing*. Com a utilização de ferramentas de monitorização que permitam determinar e analisar a utilização da solução proposta, poderá ser possível verificar se a nova estrutura está a fazer diferença relativamente à sobrecarga do OpenNebula.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. A. A. Brazeta, "Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana," em *Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana*, 2014, pp. 2-3;9-10;44-45;57.

[2] "Team Foundation Server," Microsoft, 2013. [Online]. Available: https://msdn.microsoft.com/en-us/vstudio/ff637362.aspx. [Acedido em 16 Setembro 2014].

[3] J. Sutherland e K. Schwaber, "What is Scrum?," Scrum Guides, 2010-2015. [Online]. Available: http://www.scrumguides.org/. [Acedido em 19 Janeiro 2015].

[4] M. D. Network, "Installing and Configuring Test Agents and Test Controllers," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/dd648127(v=vs.110).aspx. [Acedido em 27 Setembro 2014].

[5] O. Project, "OpenNebula," OpenNebula, 2002-2015. [Online]. Available: http://opennebula.org/. [Acedido em 15 Outubro 2014].

[6] M. Kawalerowicz e C. Berntson, "CI and yout development process," em *Continuous Integration in .NET*, Stamford, Manning, 2011, pp. 5-6.

[7] M. Kawalerowicz e C. Berntson, "What does it mean to integrate continuously?," em *Continuous Integration in .NET*, Stamford, Manning, 2011, p. 7.

[8] M. Fowler, "Continuous Integration," p. 1, 01 Maio 2006.

[9] P. Duvall, S. Matyas e A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.

[10] E. Blankenship, M. Woodward, G. Holliday e B. Keller, "Professional Team Foundation Server 2012," em *Professional Team Foundation Server 2012*, Indianapolis, John Wiley & Sons, Inc, 2013, pp. 3-4; 29-31; 495-500.

[11] M. Mason, em *Pragmatic Guide to Subversion*, Pragmatic Programmers, LLC, 2010, pp. 9-12; 18-19.

[12] T. Swicegood, "Introduction," em *Pragmatic Guide to Git*, Pragmatic Programmers, LLC., 2010, pp. 9-16.

[13] "TortoiseGit - The coolest Interface to Git Version Control," TortoiseGit, 2013. [Online]. Available: https://code.google.com/p/tortoisegit/. [Acedido em 19 Janeiro 2015].

[14] M. Kawalerowicz e C. Berntson, "Choosing the right CI server," em *Continuous Integration in .NET*, Stamford, Mannig, 2011, pp. 91-92.

[15] J. Ehn e T. Sandstrøm, "So, what is Team Foundation Server 2012?," em *Team Foundation Server 2012 Stater*, Birmingham, Packt Publishing, 2012, pp. 1-2.

[16] "About CruiseControl.NET (abbr. CCNet)," CruiseControl.NET, 2011. [Online]. Available: http://www.cruisecontrolnet.org/projects/ccnet/wiki/About. [Acedido em 19 Janeiro 2015].

[17] J. F. Smart, "Introducing Jenkins," em *Jenkins:The Definitive Guide*, Sebastopol,, O'Reilly Media, 2011, pp. 1-3.

[18] A. M. Berg, "Preface," em *Jenkins Continuous Integration Cookbook*, Birmingham, Packt Publishing, 2012, pp. 1-2; .

[19] M. Kawalerowicz e C. Berntson, "Automating the build process," em *Continuous Integration in .NET*, Stamford, Manning, 2011, pp. 62-88.

[20] E. Blankenship, M. Woodward, G. Holliday e B. Keller, "Overview of Build Automation," em *Professional Team Foundation Server 2012*, Indianapolis, John Wiley & Sons, Inc, 2013, pp. 377-380.

[21] J. Rossberg e M. Olausson, Pro Application Lifecycle Management with Visual Studio 2012, Apress, 2012.

[22] B. Soninsky, Clound Computing Bible, Indianapolis: Wiley Publishing, Inc, 2011.

[23] O. Project, "An Overview of OpenNebula," OpenNebula, 2002-2015. [Online]. Available:

http://docs.opennebula.org/4.8/design_and_installation/building_your_cloud/intro.html. [Acedido em 05 Janeiro 2015].

[24] D. M. Aranda, “An Introduction to Cloud Computing with OpenNebula,” 19 Novembro 2013. [Online]. Available: http://www.slideshare.net/opennebula/tecni-28445050. [Acedido em 10 Fevereiro 2015].

[25] O. Project, “Integration,” OpenNebula, 2002-2015. [Online]. Available: http://docs.opennebula.org/4.8/integration/getting_started/introapis.html. [Acedido em 23 Janeiro 2015].

[26] O. Project, em *OpenNebula 4.10 Design and Installation Guide, Release 4.10.2*, 2015, p. 1;5;26.

[27] C. Cook, “XML-RPC.NET,” 17 Abril 2011. [Online]. Available: http://xml-rpc.net/. [Acedido em 10 Fevereiro 2015].