****

Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Projeto e Seminário

***Plataforma de Integração Contínua***

Pavel Egorov

Iurie Marcinschi

Orientador

Doutor Porfírio Pena Filipe

Julho 2014

Resumo

No contexto de construção de *Software*, normalmente, o desenvolvimento de cada membro da equipa traduz-se em múltiplas integrações de código por dia, em que cada integração é verificada por meio de testes unitários. Assim sendo, a integração contínua é uma prática que desempenha um papel indispensável no dia a dia de um programador.

O presente projeto propõe uma plataforma *Web* que oferece suporte para facilitar a prática de integração contínua, disponibilizando ambientes *Linux* para compilação de código e execução dos testes unitários.

O programador terá possibilidade de importar o código dos seus repositorios para poder testar em ambientes *Linux* à medida, que a plataforma oferece.

Um teste na plataforma consiste em compilar o código e executar testes unitários escritos pelo programador de forma automática, disponibilizando-lhe o log da execução e o resultado.

**Sumário**

[Capítulo 1 v](#_Toc395641559)

[Introdução v](#_Toc395641560)

[Motivação v](#_Toc395641561)

[Objetivo v](#_Toc395641562)

[Contribuição v](#_Toc395641563)

[Organização do documento v](#_Toc395641564)

[Capítulo 2 v](#_Toc395641565)

[Estado da Arte v](#_Toc395641566)

[Ferramentas e Tecnologias v](#_Toc395641567)

[Capítulo 3 v](#_Toc395641568)

[Proposta de Solução v](#_Toc395641569)

[Arquitetura v](#_Toc395641570)

[Aplicações Realizadas v](#_Toc395641571)

[Capítulo 4 v](#_Toc395641572)

[Resultados Experimentais v](#_Toc395641573)

[Validação v](#_Toc395641574)

[Cenários de Demonstração v](#_Toc395641575)

[Capítulo 5 v](#_Toc395641576)

[Conclusão e Trabalho Futuro v](#_Toc395641577)

Índice de Ilustrações

[Figura 1 - Arquitetura conceptual vii](#_Toc390800402)

[Figura 2 – Diagrama de interação entre User Agent, Web Server e Redis viii](#_Toc390800403)

[Figura 3 – Diagrama de interação entre as entidades Worker ix](#_Toc390800404)

1 Introdução

O presente capítulo contextualiza o projeto apresentando a motivação e os objetivos funcionais.

1.1 Enquadramento e Motivação

Há poucas décadas atrás, equipas de desenvolvimento de Software desenvolviam código das aplicações individualmente até chegar a hora de integrar num produto. Na fase de integração, as mudanças no código de dezenas ou centenas de programadores seriam fundidas em uma base de código comum o que na maioria das vezes originava conflitos e erros durante a compilação.

Felizmente, aprendeu-se que, integrar com mais frequência evita-se o impacto com a base de código comum. Na década de 90 a compilação diária do código tornou-se uma prática normal e no início dos anos 2000 este princípio levou-se ao extremo: a integração contínua e a validação destas integrações com uma construção rápida e casos de teste.

Às vezes é necessário testar o mesmo código em ambientes com propriedades diferentes, o que se traduz em gasto de tempo nas configurações dos ambientes ou gastos financeiros adicionais para preparação/aquisição dos mesmos.

Neste contexto surge a necessidade de existir um sistema, com uma simples interface de configuração disponível na Internet, que disponibilizará recursos necessários de forma rapidamente e simples, sempre prontos em realizar testes de forma automática a medida que este é alterado no repositório *Git*, desta forma poupando tempo do programador na execução dos testes, na obtenção de recursos e configuração dos mesmos.

1.2 Objetivos

Com este projeto pretende-se conceber e implementar um sistema aplicacional *Web* que ofereça ambientes *Linux* sem interface gráfica, facilmente configuráveis para construção (*build*) e execução automática de testes.

Assim sendo o utilizador deixa de perder o seu tempo e recursos da sua máquina ou da empresa para montar um ambiente próprio e testar o seu código.

Com uma simples e fácil configuração no nosso sistema, o utilizador terá sempre recursos necessários prontos para realização dos seus testes de forma automática, deixando de executa-los manualmente, pois o sistema fará isto por ele cada vez que detetar alterações no código.

Assim sendo, o utilizador deve ter possibilidade em criar um ou mais projetos especificando as definições do ambiente necessário para executar o seu código, como por exemplo necessidade de ter uma máquina Linux com suporte para PHP, e especificando repositório *Git* que contém o código.

O sistema deve usar *Git Hooks* para ser notificado sempre que haverá alterações no código do repositório *Git* e cada vez que notificado, automaticamente descarrega-o para ser compilado e testado.

O sistema deve disponibilizar o resultado da execução e os *logs* com informaçãotanto no portal do sistema como por correio eletrónico, permitindo desta maneira o mais rapidamente possível encontrar e notificar acerca dos supostos erros nas alterações recentes do código.

Como praticamente toda aplicação *Web*, o sistema deve possuir procedimentos de registo e autenticação local e como opção via contas existentes *GitHub* dos utilizadores. Também como já referido disponibilizará a criação/configuração/eliminação dos projetos, execuções de testes, visualização da execução, recolha dos resultados e visualização do histórico das execuções. O sistema deve permitir ao utilizador criar vários projetos configurados para deferentes repositórios, dando-lhe desta maneira a possibilidade de testar código isoladamente.

Organização do documento

Estado da Arte

Este capítulo tem como objetivo apresentar as tecnologias e os conceitos básicos que suportam o trabalho realizado.

Ferramentas e Tecnologias

Todas as ferramentas descritas abaixo são de código aberto (*Open Source*).

**VAGRANT**

Para garantir que os ambientes de desenvolvimento sejam idênticos nos membros da equipa, utilizou-se a ferramenta *VAGRANT* que é basicamente um gestor para máquinas virtuais. No ficheiro de configuração *Vagrantfile* descreve-se o tipo de máquina a utilizar (exemplo *Ubuntu-amd64 bits*), as aplicações a instalar como e a forma de acesso ao ambiente. Desta forma garante-se que o aprovisionamento das ferramentas e dependências seja automático e equivalente em todas as estações de trabalho onde está a ser desenvolvido o projeto.

<https://docs.vagrantup.com/v2/>

O aprovisionamento (*Provisioning*) neste contexto significa instalar e configurar as aplicações necessárias para desenvolvimento dentro da máquina virtual, para que esta esteja pronta para o lançamento e trabalho. De outra forma dizendo, em vez de instalar e configurar manualmente as aplicações como *NodeJs, Docker, MySql, Redis* dentro da máquina virtual, optou-se em utilizar a ferramenta *CHEF* que o *VAGRANT* suporta no seu processo de aprovisionamento.

**CHEF**

É um dos sistemas mais populares de gestão de configurações em máquinas Linux. É usado para simplificar a tarefa de configuração e manutenção de servidores de uma empresa, e pode se integrar com plataformas baseadas em nuvem, como *Amazon EC2*, *Google Cloud*, *Microsoft Azure* entre outras, para provisionar automaticamente e configurar novas máquinas.

O *CHEF* funciona a base de "receitas" que são ficheiro escritos em linguagem de programação *Ruby* interpretados pelo *CHEF*, e que descrevem a gestão das aplicações e como elas devem ser configuradas. *CHEF* garante que cada recurso está devidamente configurado e corrige se for necessário.

**DOCKER**

Para criar e gerir ambientes isolados para execução de aplicações, optou-se pela ferramenta *DOCKER*. Esta ferramenta permite executar um ou mais sistema (s) operativo (s) Linux dentro de um sistema operativo Linux hospedeiro. Para este efeito, o *DOCKER*, usa um recurso do sistema operativo Linux chamado *LXC - Linux Containers* que são uma espécie de contentores (ambientes virtuais) que possuem próprio CPU, memória, I/O, rede, espaço etc. fornecidos pelo *Karnel* do *SO* *Linux* hospedeiro.

<https://linuxcontainers.org/>

<https://www.docker.com/whatisdocker/>

**NODEJS**

A plataforma de programação escolhida é *NODEJS.* É uma plataformaassente na linguagem *JavaScript* com natureza totalmente assíncrona que fornecer funcionalidades amigáveis para construção de aplicações web com carater escalável. Característica dominante está no uso de eventos I/O assíncronos não bloqueantes.

<http://nodejs.org/documentation/>

**REDIS**

É um sistema de armazenamento de dados em pares chave-valor, oferecendo algumas estruturas de dados diferentes como *strings*, *hashes*, *lists*, *sets* and *ordered sets*. Cada um tipo de estrutura tem características únicas e suporta comandos únicos. Uma das características relevantes para este projeto é *Redis* possuir funcionalidades de Publicação/Subscrição em canais de troca de dados “messaging*”* onde todo o interessado pode publicar messagens e todo o interessado pode ler.

**MYSQL**

É uma das mais populares aplicações de código aberto (*Open Source*) de base de dados relacionais.

<http://redis.io/topics/introduction>

<http://redis.io/documentation>

<http://en.wikipedia.org/wiki/Redis>

Capítulo 3

Proposta de Solução

Arquitetura

Aplicações Realizadas

Capítulo 4

Resultados Experimentais

Validação

Cenários de Demonstração

Capítulo 5

Conclusão e Trabalho Futuro

1 Introdução

O presente capítulo contextualiza o projeto, apresenta os objetivos e as funcionalidades implementadas no projeto.

* 1. Enquadramento e Motivação

Há muito tempo atrás, equipas de desenvolvimento de *Software* desenvolviam os recursos individualmente até chegar a hora de integrar num produto. Na fase de integração, as mudanças no código de dezenas ou centenas de programadores seriam fundidas em uma base de código comum o que na maioria das vezes originava conflitos durante a compilação.   
Felizmente, aprendeu-se que, integrar com mais frequência evita-se o impacto com o base de código comum. Na década de 90 a compilação diária do código tornou-se uma prática normal e no início dos anos 2000 este princípio levou-se ao extremo: a integração contínua e a validação destas integrações com uma construção rápida e casos de teste.

Neste contexto surge a necessidade de ter sempre a mão um ambiente que reúne características necessárias para efetuar a construção do código e execução dos casos de teste.

Muitas vezes é necessário testar o código em ambientes com propriedades diferentes, o que se traduz em gasto de tempo ou gastos financeiros adicionais para preparação/aquisição dos mesmos.

1.2 Objetivo

Com este projeto pretende-se conceber e implementar um sistema que ofereça ambientes configuráveis para construção e execução de testes de forma fácil e rápida.

O sistema tem seguinte utilização:

1. Registar uma conta de utilizador
2. Permitir autenticação via GitHub
3. Criar projeto(s):
   1. Seleção da linguagem do código a ser executado (ex.: Node.js, PHP)
   2. Seleção das dependências (ex.: Redis)
   3. Comandos a ser executados sobre o código.
4. Atualizar ou eliminar projeto(s)
5. Efetuar múltiplas execuções do código do mesmo projeto.
6. Notificação por *e-mail* do utilizador com o resultado da execução.
7. Visualizar os resultados da execução
8. Visualizar os *Logs* da execução
9. Visualizar o histórico de todas execuções de um determinado projeto.

2 Estado da arte

Este capítulo tem como objetivo apresentar as tecnologias e os conceitos básicos que suportam o trabalho realizado.

2.1 Arquitetura conceptual

A arquitetura conceptual do sistema tem dois componentes principais *Worker* e *Web* que comunicam entre si via Redis.

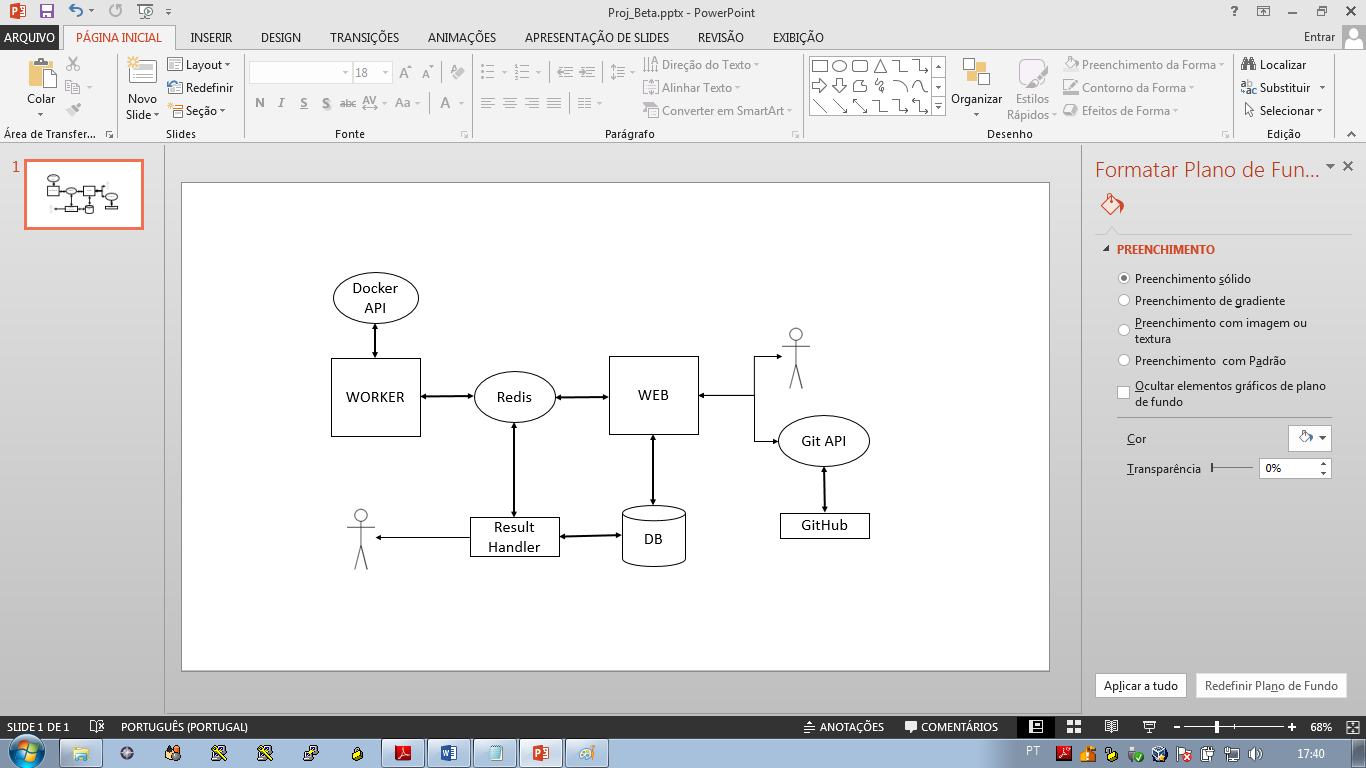


Figura 1 - Arquitetura conceptual

2.2 Componente Web

A componente Web é um servidor *http* que disponibiliza interface para utilizadores que interagem com a plataforma.

Implementada com base na *framework ExpressJs* que segue o padrão *MVC (Model View Controller)*. Usa base de dado relacional *MySql* para armazenamento persistente dos dados.

2.2.1 Interação do Utilizador e Servidor Web

A plataforma oferece visualização do progresso da execução de um *Build* exemplificado na *Figura 3*.

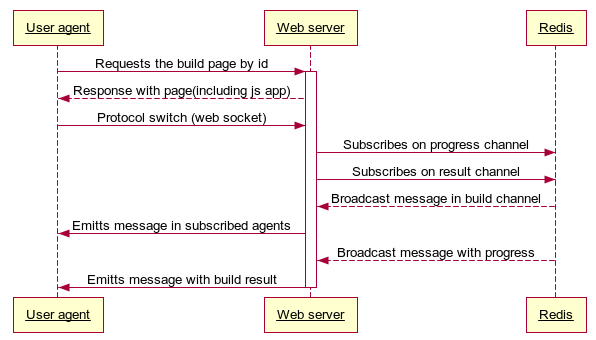


Figura 2 – Diagrama de interação entre User Agent, Web Server e Redis

2.3 Componente Worker

Componente Worker é constituída quatro entidades *Worker Client*, *Redis*, *Worker* e *Docker*.

O principal objetivo do *Woker Client* é processar os *trabalhos* (*Builds*) e devolver os resultados a fila de resultados, como também publicar o progresso para o canal *publish/subscribe* do Redis.

O *Worker* trata de controlar e configurar a execução de um *Bulid* dentro de um Contentor fornecido pelo *Docker*.

O *Docker* é uma aplicação que permite execução de aplicações e a gestão das mesmas dentro de um ambiente virtual a nível do sistema operativo *LXC* (userspace *interface* para contenção dos recursos *Kernel Linux*). Permite "empacotar" a aplicação com todas as dependências em um recipiente (Contentor), bem como fornece um ambiente para o gerenciamento de contentores.

O *Redis* é uma base de dados não relacional que oferece modelo de dados em forma de par chave/valor. Também fornece funcionalidades para implementar estruturas de dados tais como listas, conjuntos ordenados e tabelas de *Hash* . Também possui mecanismo de troca de mensagens utilizando o padrão *publish – subscribe*.

2.3.1 Interação entre as entidades do Worker

Tal como exemplificado na *Figura 2*, a interação entre entidades é:

1. *Worker Client* espera pelo trabalho registando-se na fila de *Builds* do *Redis*
2. *Redis* notifica o *Worker Client* acerca de um trabalho.
3. *Worker Client* manda o *Worker* processar o trabalho.
4. *Worker* devolve ao Worker *Client* o objeto que representa o estado do processamento do trabalho.
5. *Worker* pede uma instância do contentor via *Docker*.
6. *Docker* devolve um contentor ao *Worker*.
7. *Worker* executa o *build script* dentro do Contentor.
8. *Docker* ao longo de execução do trabalho escreve para *Standart* *Output Stream no que o Worker* está a escuta.
9. *Worker* agrega os dados obtidos do *Standart* *Output Stream* e emite evento progresso para o *Worker Client*:
10. *Worker Client* ao receber o evento “*progress*” publica no canal *Publish/Subscribe* do *Redis*
11. *Docker* informa o *Worker* da conclusão do trabalho.
12. *Worker* emite o evento “*complete*” para o *Worker Client*.
13. Worker Client por sua vez coloca o resultado na fila dos resultados do Redis.

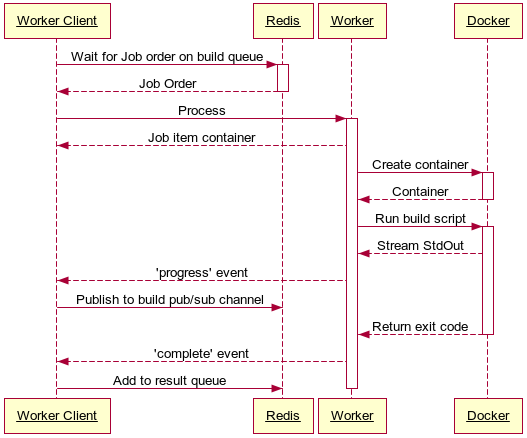


Figura 3 – Diagrama de interação entre as entidades Worker