****

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 repositórios 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.

# Agradecimentos

Queremos agradecer a todos que de alguma forma contribuíram para que este projeto fosse possível, e um particular agradecimento ao Professor Doutor Porfírio Pena Filipe por ter aceitado orientar o projeto.

# Índice Figura

[Figura 1 – Ferramentas e Tecnologias 3](#_Toc397458641)

[Figura 2 - Modelo antes da solução 6](#_Toc397458642)

[Figura 3 - Modelo depois da solução 7](#_Toc397458643)

[Figura 4 - Arquitetura geral 8](#_Toc397458644)

# Índice Tabela

**Sumário**

[Resumo i](#_Toc397532348)

[Agradecimentos ii](#_Toc397532349)

[Índice Figura iii](#_Toc397532350)

[Índice Tabela iv](#_Toc397532351)

[1 Introdução 1](#_Toc397532352)

[1.1 Enquadramento e Motivação 1](#_Toc397532353)

[1.2 Objetivos 2](#_Toc397532354)

[1.3 Organização do documento 3](#_Toc397532355)

[2 Estado da Arte 4](#_Toc397532356)

[2.1 Ferramentas e Tecnologia 4](#_Toc397532357)

[2.2 Fluxo do desenvolvimento 6](#_Toc397532358)

[3 Solução e Arquitetura 8](#_Toc397532359)

[3.1 Solução proposta 8](#_Toc397532360)

[3.2 Arquitetura Geral 9](#_Toc397532361)

[4 Implementação 11](#_Toc397532362)

[4.1 Worker 11](#_Toc397532363)

[4.2 Hub 11](#_Toc397532364)

[4.3 Web 11](#_Toc397532365)

[5 Resultados Experimentais 12](#_Toc397532366)

[5.1 Resultados 12](#_Toc397532367)

[5.1 Validação 12](#_Toc397532368)

[5.3 Cenários de Demonstração 12](#_Toc397532369)

[6 Conclusão e Trabalho Futuro 12](#_Toc397532370)

[7 Referencias 12](#_Toc397532371)

# 1 Introdução

O presente capítulo contextualiza e apresenta a motivação e os objetivos a alcançados.

## 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 e testar com mais frequência evita-se o impacto na 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.

Exemplo cenário:

1. O programador A e B descarregaram uma cópia do código comum na mesma altura.
2. O programador A cria uma classe C2 que faz uso da classe C1.
3. O programador B altera o código da classe C1 adicionando uma dependência.
4. Os dois programadores testam o código localmente nas suas máquinas com as cópias de base de código comum inicialmente descarregadas.
5. Depois de os testes localmente efetuados forem bem-sucedidos, submetem as alterações para o repositório de base de código comum.

Neste exemplo demonstra-se como as alterações feitas na classe C1 não são consideradas nos desenvolvimentos do programador A, até que alguém testar todas as alterações feitas até um determinado ponto e detetar o erro.

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 sobre o código comum, 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.

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

Assim sendo, o utilizador deve ter possibilidade de criar um ou mais projetos, especificando as definições do ambiente necessário para execução do código, como por exemplo necessidade de ter uma máquina Linux com suporte para PHP, e especificar o 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 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 deve possuir funcionalidade para 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 e se for necessário em ambientes com propriedades diferentes, como por exemplo ambientes com versões Java diferentes.

## 1.3 Organização do documento

Este documento encontra-se organizado nos capítulos:

1. Capítulo 1 – Introdução: capítulo atual. Contextualiza e apresenta a motivação e os objetivos a alcançar.
2. Capítulo 2 - Estado da arte: apresenta as tecnologias e os conceitos básicos que suportam o trabalho realizado.
3. Capítulo 3 - Solução e arquitetura: apresenta a solução proposta e arquitetura geral do projeto, descrevendo os seus componentes e a interação entre os mesmos.
4. Capítulo 4 – Implementação: descreve a implementação das componentes que integram a solução apresentada no Capítulo 3.
5. Capítulo 5 – Conclusão: apresenta a análise crítica sobre o projeto e as limitações do mesmo assim como possíveis desenvolvimentos futuros.

# 2 Estado da Arte

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

## 2.1 Ferramentas e Tecnologia

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

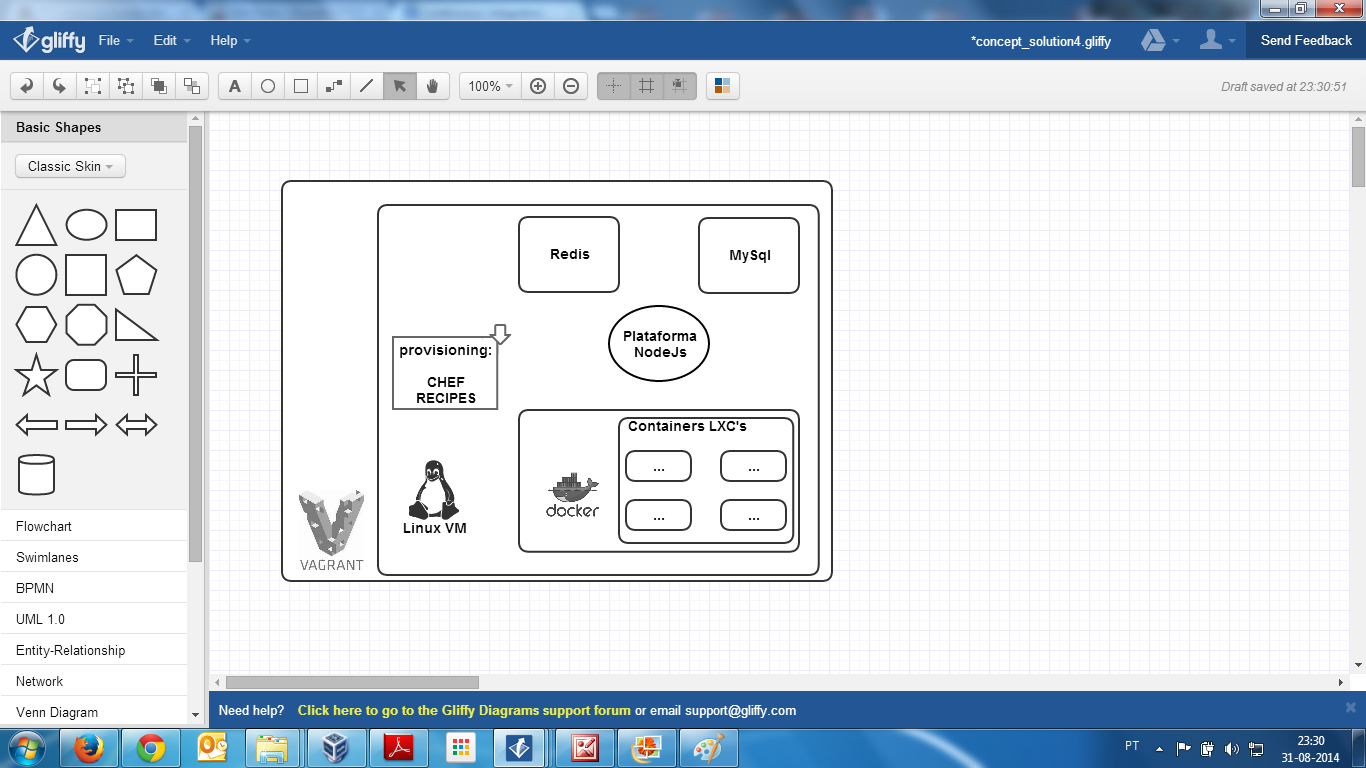


Figura – Ferramentas e Tecnologias

**VirtualBox**

É um *software* de virtualização que criar ambientes para instalação e utilização de um ou mais sistemas operativos dentro do sistema operativo da máquina física, compartilhando desta maneira o mesmo hardware.

**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 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,* em que se descrevem de forma programática a gestão das aplicações e como elas devem ser configuradas.

As receitas são agrupadas em coleções chamados *cookbook*. A *cookbook* é uma unidade fundamental de configuração e de políticas de distribuição. Cada *cookbook* define um cenário, como por exemplo, define tudo o que é necessário (dependências) para instalação e configuração do MySql, e depois contem todos os componentes que são obrigatórios para suportar o tal cenário.

*Chef* garante que cada recurso está devidamente configurado.

<http://en.wikipedia.org/wiki/Vagrant_(software)>

<https://wiki.opscode.com/display/chef/Home>

<https://docs.getchef.com/essentials_cookbooks.html>

**Berkshelf**

Berkshelf é um gestor de dependências para o Chefe, aprovisiona o Chefe com livros de receitas focados para um determinado componente, reutilizável e configurável. Berkshelf encara os livros de receitas como bibliotecas de aplicações.

**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/>

**MOCHA**

Tal como *JUnit* para *Java* e *NUnit* para *Microsoft .NET*, *Mocha* em plataformas *NodeJS* é uma *framework* de teste para código *JavaScript*.

<http://visionmedia.github.io/mocha/>

**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 mensagens e todo o interessado pode ler.

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

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

<http://redis.io/documentation>

**MYSQL**

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

<http://pt.wikipedia.org/wiki/MySQL>

## 2.2 Fluxo do desenvolvimento

O recurso fundamental do sistema a desenvolver é o sistema operativo *Linux*, assim sendo, a primeira necessidade que surgiu foi montar um emulador para o mesmo. Por ser já reconhecido e o mais utilizado foi escolhido o emulador “*Oracle VM VirtualBox”*.

O próximo passo foi investigar como é que podíamos garantir que todas as alterações efetuadas no ambiente de desenvolvimento sejam facilmente replicadas em todas as estações de trabalho em que o projeto é desenvolvido e como ter um gestor que permite gerir a máquina virtual Linux sem interface gráfica a partir do ambiente de trabalho da máquina hospedeira, oferecendo a possibilidade em desenvolver o código a partir da máquina host e executa-lo na máquina virtual. A solução encontrada foi o gestor de máquinas virtuais *Vagrant*. O *Vagrant* permite configurar o mapeamento de portos de encaminhamento da máquina virtual para serem acedidos a partir dos portas específicos da máquina hospedeira, permite também configurar o aprovisionamento automático da máquina virtual com ferramentas necessárias para o funcionamento da aplicação. Assim sendo para manter os ambientes de execução da aplicação idênticos basta partilhar o ficheiro de configuração *vagrantfile.*

Ao executar o comando “*vagrant up*” é executado o “*start”* da máquina virtual.

Via comando “vagrant provision” é lançado o processo de aprovisionamento da mesma.

Via o comando “*vagrant ssh*” acede-se para dentro da máquina virtual tendo lá já a cópia de ficheiros e estrutura das pastas do projeto da máquina *host*.

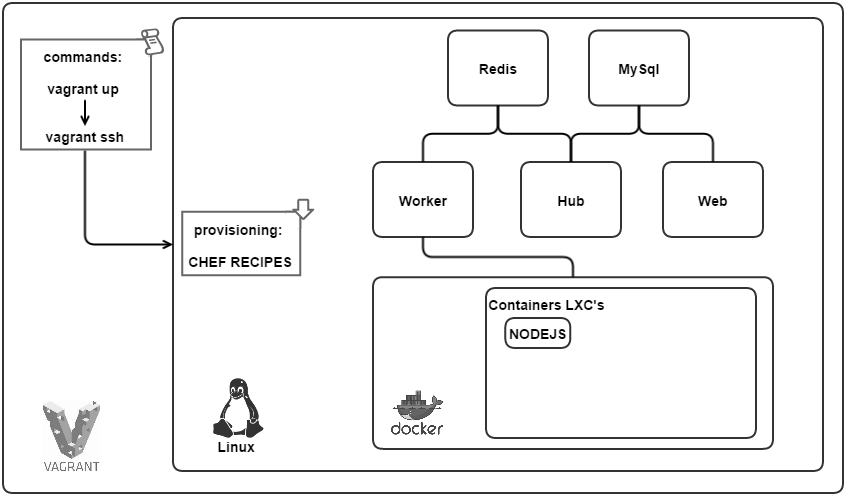


Figura - Ambiente de desenvolvimento

# 3 Solução e Arquitetura

Este capítulo apresenta a solução proposta e arquitetura geral do projeto, descrevendo os seus componentes e a interação entre os mesmos.

## 3.1 Solução proposta

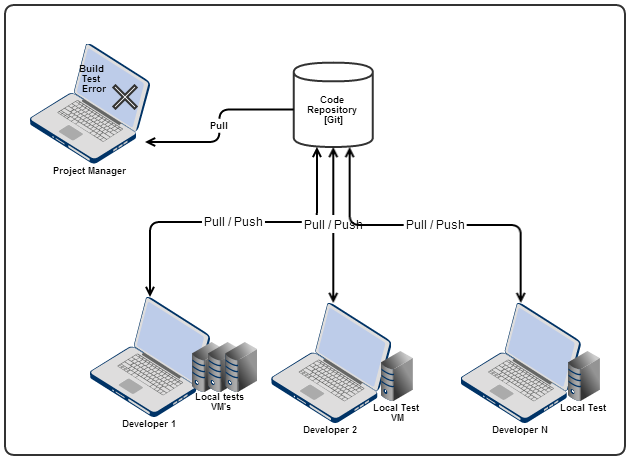


Figura - Modelo antes da solução

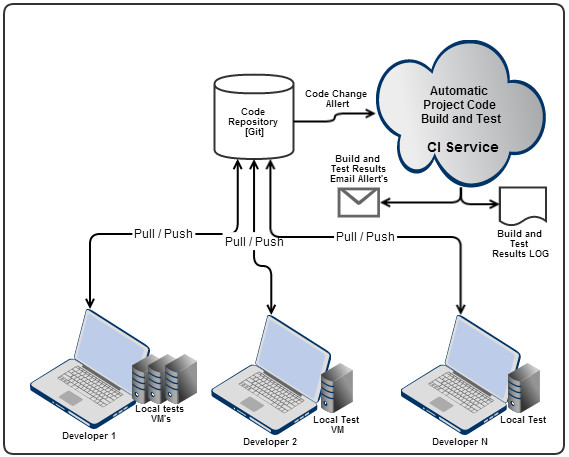


Figura - Modelo depois da solução

## 3.2 Arquitetura Geral

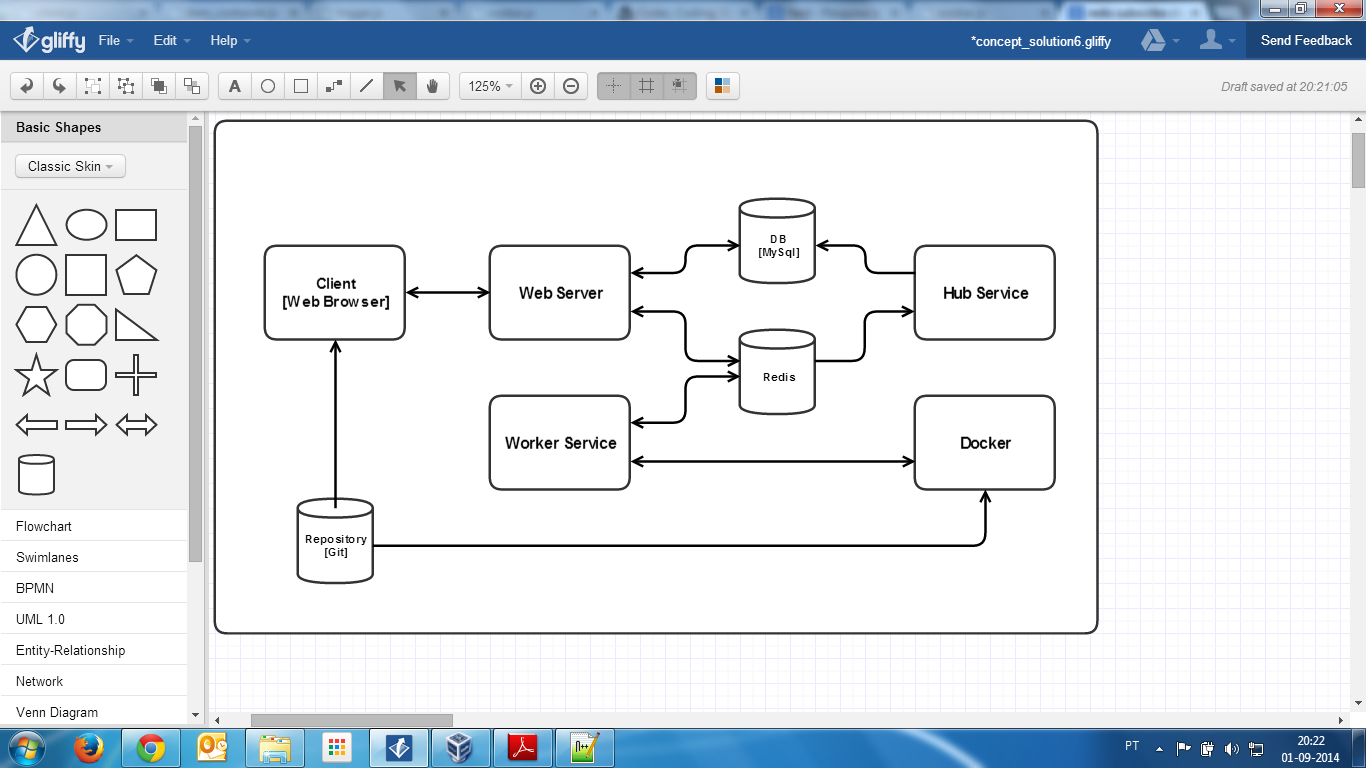


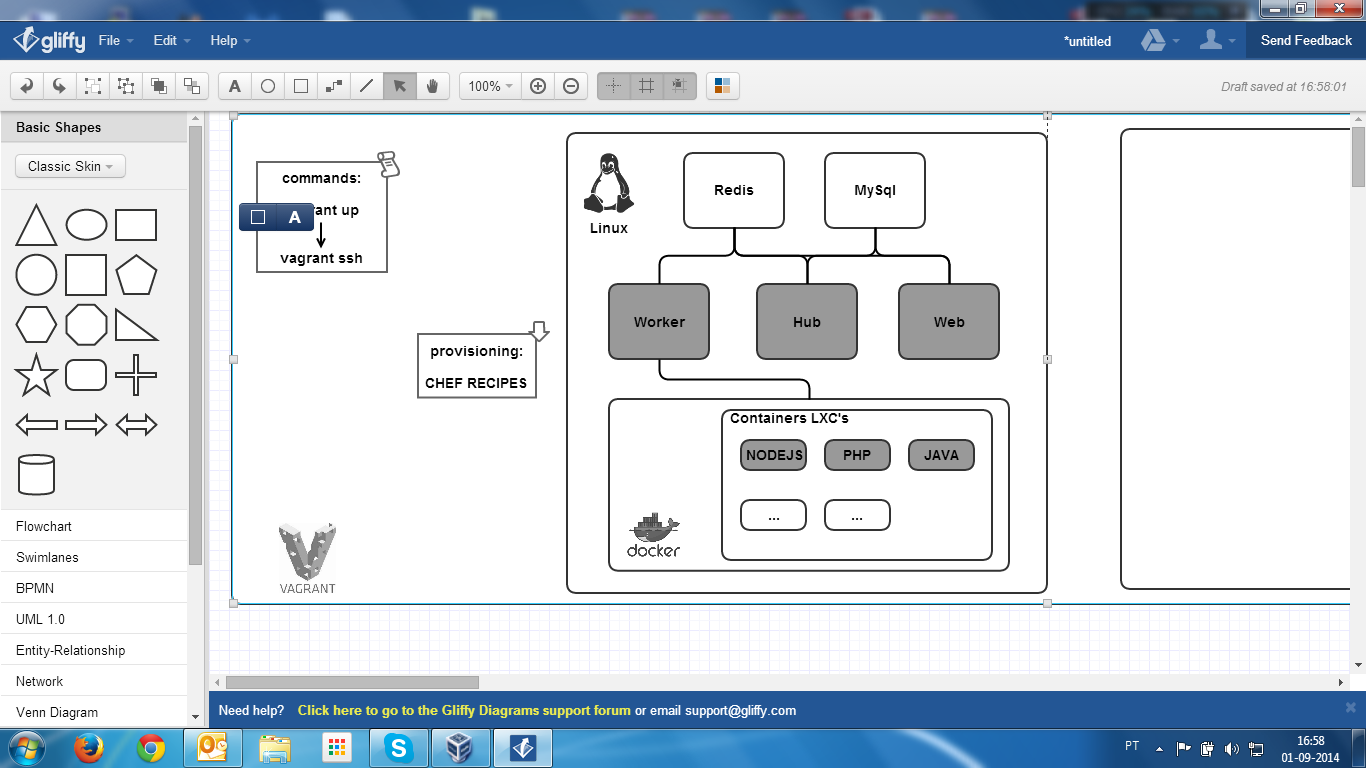
Figura - Arquitetura geral

# 4 Implementação

## 4.1 Worker

## 4.2 Hub

## 4.3 Web



# 5 Resultados Experimentais

## 5.1 Resultados

## 5.1 Validação

## 5.3 Cenários de Demonstração

# 6 Conclusão e Trabalho Futuro

# 7 Referencias

[1] - <http://en.wikipedia.org/wiki/Continuous_integration>