LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE – UNI-RN
ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS CORPORATIVOS

EDER BRENDO DA SILVA SOUZA

IMPLEMENTAÇÃO DE UM AMBIENTE CLUSTERIZADO COM ALTA DISPONIBILIDADE PARA SISTEMAS JAVA WEB UTILIZANDO DOCKER

NATAL - RN
2017

LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE – UNI-RN
ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS CORPORATIVOS

EDER BRENDO DA SILVA SOUZA

IMPLEMENTAÇÃO DE UM AMBIENTE CLUSTERIZADO COM ALTA DISPONIBILIDADE PARA SISTEMAS JAVA WEB UTILIZANDO DOCKER

Trabalho de Conclusão de Curso apresentado ao Centro Universitário do Rio Grande do Norte, em cumprimento às exigências legais como requisito final para obtenção do título de Especialista em desenvolvimento de Sistemas Corporativos.

Orientador: Prof. Me. Romulo Fagundes Cantanhede

NATAL - RN
2017

EDER BRENDO DA SILVA SOUZA

IMPLEMENTAÇÃO DE UM AMBIENTE CLUSTERIZADO COM ALTA DISPONIBILIDADE PARA SISTEMAS JAVA WEB UTILIZANDO DOCKER

Trabalho de Conclusão de Curso apresentado ao Centro Universitário do Rio Grande do Norte, em cumprimento às exigências legais como requisito final para obtenção do título de Especialista em Desenvolvimento de Sistemas corporativos.

Aprovado (a) em 04/10/2017

BANCA EXAMINADORA

_____

Prof. Romulo Fagundes Cantanhede
Orientador

_____

Prof. Gleydson de Azevedo Ferreira Lima
Membro

Dedico este trabalho aos meus familiares, em especial aos meus pais e minha irmã que foram responsáveis por viabilizar essa conquista.

RESUMO

Implementar um ambiente clusterizado para sistemas desenvolvidos em Java possui uma certa dificuldade, além dos vários arquivos de configurações necessários, talvez o principal problema seja por conta da necessidade de mais de uma máquina para isso, mesmo com a virtualização, administrar essas máquinas é bastante cansativo, devido principalmente a necessidade de manter vários sistemas operacionais ativos. O ambiente proposto busca diminuir essas dificuldades utilizando o Docker, que irá agir como um facilitador para sua criação, sem a necessidade de várias máquinas virtuais, onde foram substituídas por containers. Será apresentado um ambiente, utilizando como servidor de aplicação o Wildfly, Servidor WEB Apache HTTP e SGBD PostgreSQL, onde após a implementação, foi solucionado o problema de segurança do cenário anterior e comprovado o aumento de disponibilidade da aplicação através dos testes de carga realizados nos dois ambientes.

PALAVRAS-CHAVE: Aplicação WEB, Java, Docker, Cluster, Infraestrutura, Ambiente.

ABSTRACT

Implementing a clustered environment for systems developed in Java has a certain difficulty, in addition to the various configuration files needed, perhaps the main problem is due to the need of more than one machine for this, even with virtualization, administering these machines is quite tiring , mainly due to the need to maintain multiple active operating systems. The proposed environment seeks to reduce these difficulties using Docker, which will act as a facilitator for its creation, without the need for several virtual machines, where they were replaced by containers. Will be presented an environment , such as application server Wildfly, Web server Apache HTTP and PostgreSQL DBMS, where after the implementation, the security problem of the previous scenario has been solved and proves the increased availability of the application through the load tests performed in both environments.

KEYWORDS: WEB application, Java, Docker, Cluster, Infrastructure, Environment.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

HTTP – Hypertext Transfer Protocol

JPA – Java Persistence API

JVM – Java Virtual Machine

SGBD – Sistema de Gerenciamento de Banco de Dados

SQL – Structured Query Language

WEB – World Wide Web

LXC – Linux Container

SUMÁRIO

1. INTRODUÇÃO

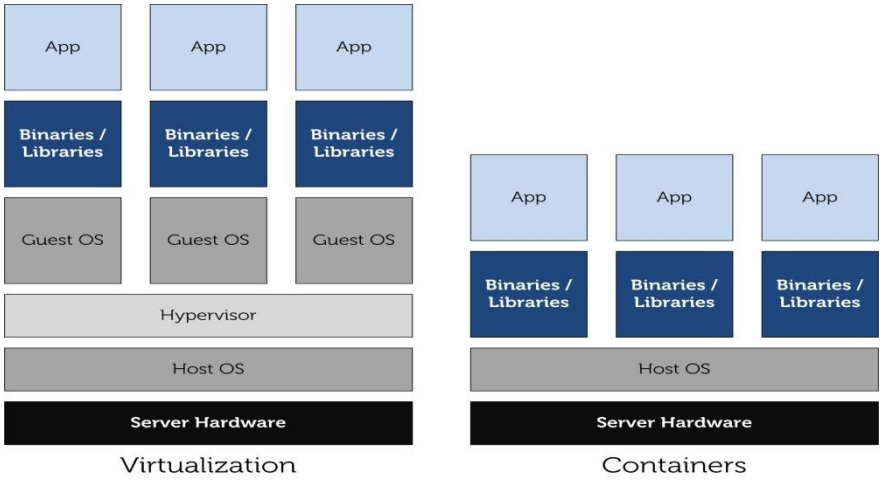
Implementar um ambiente clusterizado para sistemas desenvolvidos em Java possui uma certa dificuldade, além dos vários arquivos de configurações necessários para configurar o ambiente, talvez o principal problema seja por conta da necessidade de mais de uma máquina para isso, mesmo com a virtualização, administrar essas máquinas é bastante cansativo, devido principalmente a necessidade de manter vários sistemas operacionais ativos.

Um ambiente clusterizado para sistemas Java nada mais é que duas instâncias de um servidor de aplicação Java que hospede a mesma aplicação, com um balanceador de carga para distribuir as requisições entre eles, com isso, o ambiente se torna de alta disponibilidade, pois caso, uma das instâncias falhem, outra irá assumir sem maiores problemas, e para manter esse ambiente, no pior dos casos serão necessárias três maquinas, uma para o balanceador de carga e duas para manter o servidor de aplicação, acrescentando o banco de dados, teremos então quatro maquinas, sejam elas físicas ou virtuais.

A proposta desse trabalho então é criar um ambiente menos burocrático, onde em vez de quatro maquinas completas, sejam virtuais ou não, a intenção é ter apenas uma máquina para comportar todo o ambiente, utilizando Docker.

> Docker não é um sistema de virtualização tradicional. Enquanto em um ambiente de virtualização tradicional nós temos um S.O completo e isolado, dentro do Docker nós temos recursos isolados que utilizando bibliotecas de kernel em comum (entre host e container), isso é possível pois o Docker utiliza como backend o nosso conhecido LXC. (DIEDRICH, 2015)

Figura 1 - Virtualização x Container



Fonte: <https://www.mundodocker.com.br/o-que-e-docker/> Acesso em: 19 de Setembro de 2017

Com o Docker, será possível criar todo o ambiente, seguindo uma arquitetura multicamada, em uma única máquina, pois como disse DIEDRICH (2015), o Docker utiliza o LXC (Linux Container), que é uma forma de virtualização a nível de sistema operacional, permitindo então executar vários sistemas Linux (Containers) compartilhando um único kernel, no caso o da máquina hospedeira, além disso, com o Docker será possível criar todos os containers de forma automatizada, o Docker possui uma comunidade chamada Docker hub, onde qualquer um pode compartilhar seu ambiente, evitando então a dor de cabeça de arquivos de configuração.

O servidor de aplicação Java do ambiente proposto será o Wildfly, o motivo da escolha se deu por ser um servidor completo, e possuir um modo em que será possível administrar a aplicação de forma centralizada, onde todas as configurações serão feitas em uma única máquina, e distribuídas nas demais instâncias.

O servidor Web será o Apache HTTP, que terá como objetivo também realizar o balanceamento de carga do cluster, com um módulo chamado "mod_cluster", a vantagem é que ele já está incorporado no Wildfly, facilitando então sua configuração.

O sistema gerenciador de banco de dados será o PostgreSQL, o motivo é que além de ser open source é consolidado no mercado.

A empresa Magú Material de Construção possui um sistema desenvolvido em Grails, em um ambiente que tem como servidor de aplicação o Tomcat, onde foi verificado que através do log de informações do mesmo que

estava sofrendo tentativas constantes de acesso a sua área administrativa, via força bruta, caracterizando uma falha de segurança.

O ambiente proposto foi implementado com o objetivo de resolver esse problema de segurança e melhorar a disponibilidade da aplicação.

1.1. OBJETIVO GERAL

Apresentar um ambiente clusterizado com alta disponibilidade para sistemas web Java, utilizando Docker.

1.2. OBJETIVOS ESPECIFICOS

- Descrever o cenário que motivou a criação do ambiente;
- Implementação do ambiente;
- Testes de comparação entre os dois ambientes;

1.3. JUSTIFICATIVA

A Magú Material de Construção é uma pequena empresa de comercio de Materiais de Construções, Situada em Natal/RN, atualmente com 13 funcionários, possui uma aplicação web desenvolvida em Grails que serve como página pública para a apresentação da empresa e possui uma área restrita para controle de contas e vendas em convenio, utilizada por dois usuários.

O cenário que comporta a aplicação apresentou uma falha grave de segurança, que motivou a criação de um novo ambiente utilizando tecnologias mais recentes, buscando mais segurança, fácil implantação sem a necessidade de investimentos em hardware.

2. REFERENCIAIS TEORICOS

2.1. SISTEMAS DISTRIBUÍDOS

Segundo Coulouris et al (2007, p.15), ”Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens”.

Segundo Tanenbaum et al (2007, p.1), “Um sistema distribuído é um conjunto de computadores independentes que se apresenta aos seus usuários como um sistema único e coerente”.

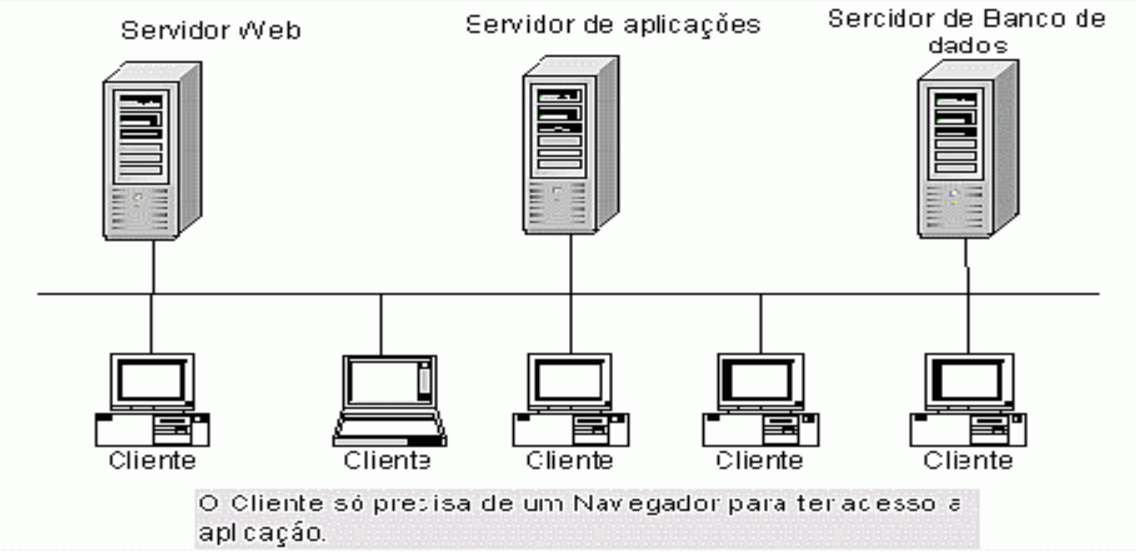
SILVA (2012 apud Tanenbaum et al, 2007), diz que a escalabilidade da estrutura é uma das características de um sistema distribuído, pois, fazendo uso da independência das unidades no comportamento do conjunto, proporciona por meio de adição de componentes um aumento da capacidade e recursos na estrutura como um todo.

2.2. MODELO ARQUITETURAL DE 4 CAMADAS

Sobre a idéia básica do Modelo arquitetural de quatro camadas, MACÊDO (2012) diz:

> [...] retirar a apresentação do cliente e centralizá-las em um determinado ponto, o qual na maioria dos casos é um servidor Web. Com isso o próprio Cliente deixa de existir como um programa que precisa ser instalado em cada computador da rede. O acesso a aplicação, é feito através de um Navegador, como o Internet Explorer ou o Netscape Navigator.

Figura 2 *- Modelo Arquitetural de 4 Camadas*



Fonte: <http://www.diegomacedo.com.br/arquitetura-de-aplicacoes-em-2-3-4-ou-n-camadas/>
Acesso em: 15 de Setembro de 2017

Conforme MACÊDO (2012), para acessar a aplicação, o cliente utiliza o seu endereço em um navegador, onde o acesso ao banco de dados é feito de acordo com as regras contidas no servidor de aplicação, impossibilitando o acesso direto do cliente ao banco de dados. As quatro camadas deste modelo são:

- **Cliente**: Navegador utilizado pelo usuário.
- **Apresentação**: Servidor Web, interface composta por páginas HTML, ASP, ou qualquer outra tecnologia capaz de gerar conteúdo para o navegador, onde as alterações na interface da aplicação são feitas diretamente nesta camada, ficando disponíveis automaticamente a todos os clientes, sem a necessidade de reinstalar a aplicação em todos os computadores da rede.
- **Lógica**: Servidor de aplicação, local onde ficam as regras de negócio que determina de que maneira os dados serão utilizados, desta forma, quando uma regra de negócio for alterada, após atualizar no servidor de aplicação, todos os usuários passarão a ter acesso à nova versão, sem a necessidade de reinstalar a aplicação em cada um dos computadores da rede.

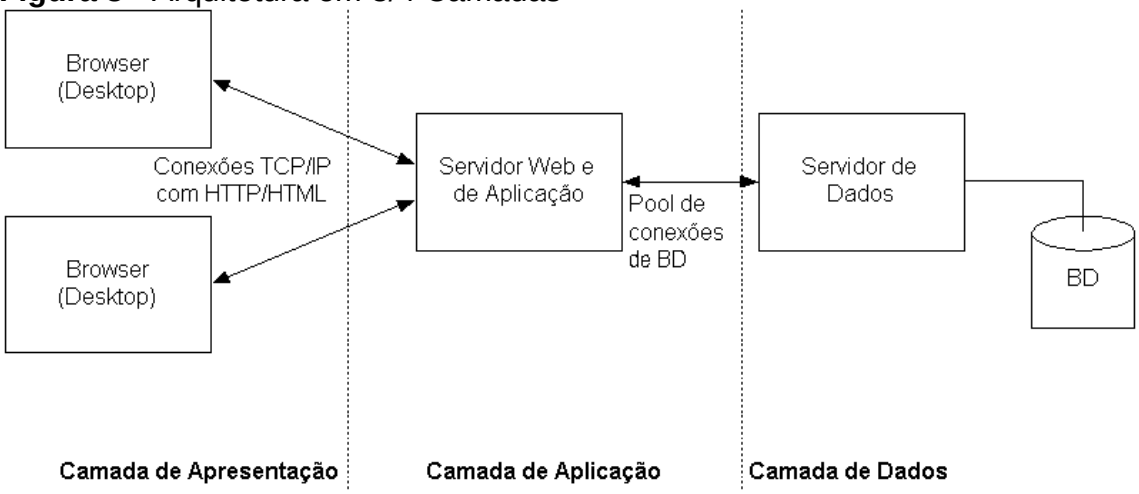
- **Persistência**: Servidor de Banco de Dados, local que reside toda a informação necessária para o funcionamento da aplicação.

MACÊDO (2012) diz que o servidor de aplicação, servidor web e servidor de banco de dados, não precisam necessariamente ser servidores separados:

> [...] uma máquina para fazer o papel de cada um dos servidores. O conceito de servidor de Aplicação, Web ou Banco de dados, é um conceito relacionado com a função que o servidor desempenha. Podemos ter, em um mesmo equipamento, um Servidor de aplicações, um servidor Web e um servidor de Banco de dados. Claro que questões de desempenho devem ser levadas em consideração.

Na plataforma Java, um exemplo de arquitetura de quatro camadas é um ambiente com container web (Servlet Container), que basicamente é um servidor de aplicação Java que possui um servidor web embutido, como mostra a figura a seguir.

Figura 3 - Arquitetura em 3/4 Camadas



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm> Acesso em: 15 de Setembro de 2017

2.3. MODELO ARQUITETURAL MULTICAMADA

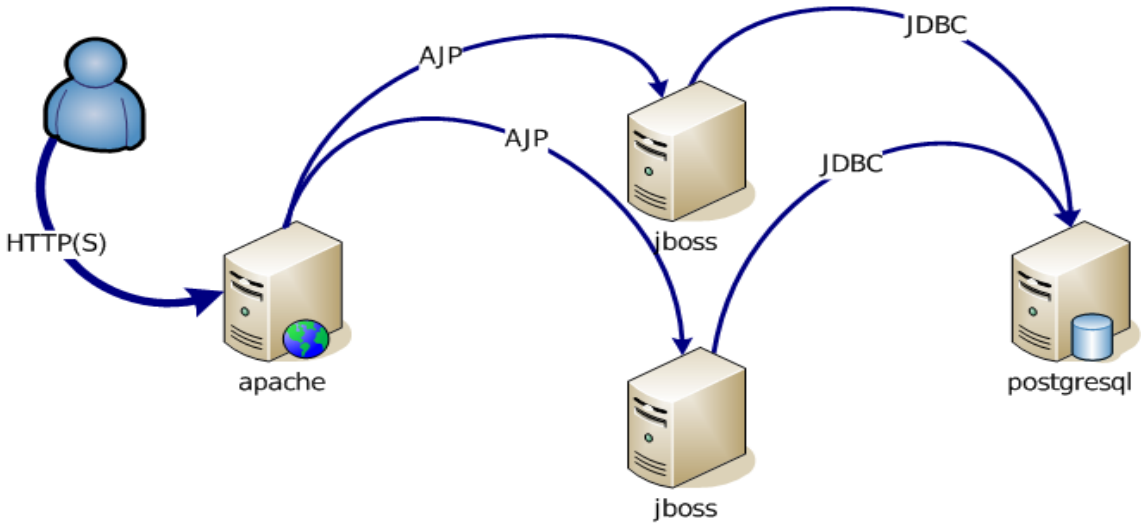
O Modelo arquitetural multicamada, contem a mesma estrutura do modelo de quatro camadas, isso é, são separadas entre o cliente, apresentação, lógica e persistência, o diferencial dessa arquitetura é a possibilidade de ter várias camadas lógicas contendo a aplicação.

MACÊDO (2012) diz que é possível implementar em um ambiente com multicamada:

- Tolerância a falhas com fail-over;
- Gerência de transações distribuídas;
- Balanceamento de carga;
- Resource pooling;
- Etc;

Na plataforma Java, comparando com a arquitetura em 3/4 camadas, a arquitetura multicamada é mais segura devido à restrição de acesso ao servidor de aplicação, onde não é mais possível acessar o mesmo diretamente pelo navegador, função realizada agora pelo servidor web, o ambiente pode ter duas ou mais instâncias do servidor de aplicação, onde o servidor web será o responsável pelo balanceamento de carga, dando uma maior disponibilidade para aplicação, pois caso um servidor falhe, o outro assume, não havendo interrupção no funcionamento, como mostra a figura a seguir:

Figura 4 - Exemplo de Modelo Arquitetural N Camadas



Fonte: <*http://rlogiacco.blogspot.com.br/2009/10/jboss-production-environment.html*> Acesso em: 15 de Setembro de 2017

2.4. CLUSTER

Sobre a definição de Cluster, o Portal OPSERVICES (2015) diz:

> Cluster é um termo em inglês que significa “aglomerar” ou “aglomeração” e pode ser aplicado em vários contextos. No caso da computação, ele define uma arquitetura de sistema capaz combinar vários computadores para trabalharem em conjunto ou, então, pode denominar o grupo em si de computadores combinados.

Ainda o Portal OPSERVICES (2015) diz:

> Cada estação é denominada “nodo” e, combinadas, formam o cluster. Em alguns casos, é possível ver referências como “supercomputadores” ou “computação em cluster” para o mesmo cenário, representando o hardware usado ou o software especialmente desenvolvido para conseguir combinar esses equipamentos.

Conforme Pitanga (2003), a forma mais básica de um cluster, é um sistema que compreende dois ou mais computadores ou sistemas, geralmente denominados de nós, que trabalham em conjunto para executar aplicações ou realizar tarefas, de forma que não fique perceptível ao usuário, criando a ilusão de um recurso único, onde é implementado o conceito de transparência de sistema. As principais características dessa plataforma é a elevação da confiança, distribuição de carga e performance.

Sobre os tipos de Clusters, Pitanga (2003) descreve:

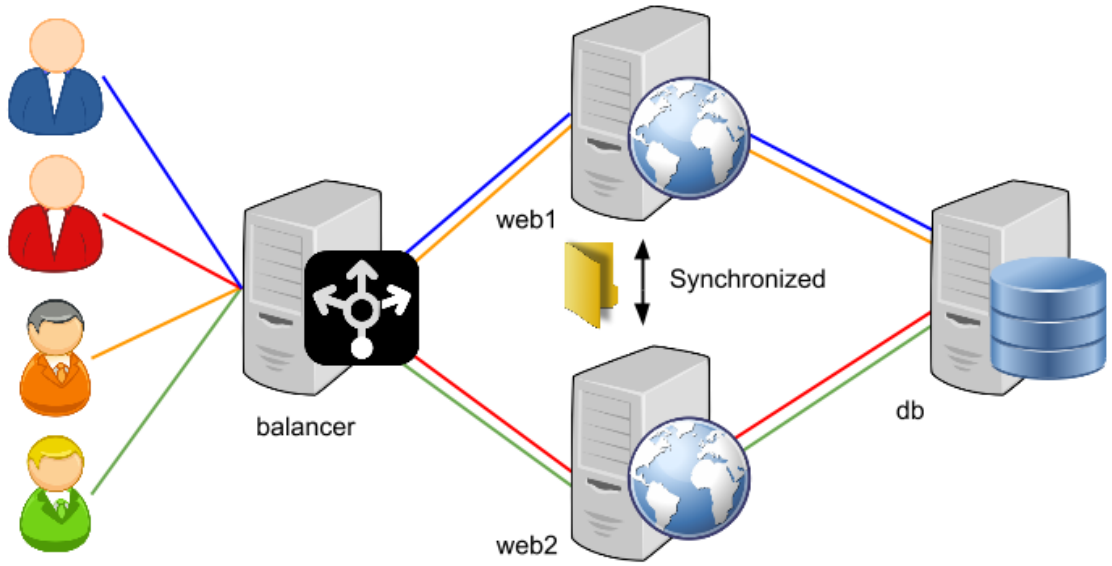
- **Alta Disponibilidade (High Availability (HA) and Failover)**, estes modelos de clusters são construídos para prover uma disponibilidade de serviços e recursos de forma ininterruptas através do uso da redundância implícitas ao sistema. A idéia geral é que se um nó do cluster vier a falhar (failover), aplicações ou serviços possam estar disponíveis em outro nó. Estes tipos de cluster são utilizados para base de dados de missões críticas, correio, servidores de arquivos e aplicações.

- **Balanceamento de carga (Load Balancing)**, este modelo distribui o tráfego entrante ou requisições de recursos provenientes dos nodos que executam os mesmos programas entre as máquinas que compõem o cluster. Todos os nodos estão responsáveis em controlar os pedidos. Se um nó falhar, as requisições são

redistribuídas entre os nós disponíveis no momento. Este tipo de solução é normalmente utilizado em fazendas de servidores de web (web farms).

- **Combinação HA & Load Balancing**, como o próprio nome diz combina as características dos dois tipos de cluster, aumentando assim a disponibilidade e escalabilidade de serviços e recursos. Este tipo de configuração de cluster é bastante utilizado em servidores de web, mail, news ou ftp.

Figura 5 - Exemplo de Cluster com a Combinação de Alta Disponibilidade e Balanceamento de Carga



Fonte: <http://www.linuxsysadmin.com.br/wp-content/uploads/2015/08/loadbalance.png>
Acesso em: 15 de Setembro de 2017

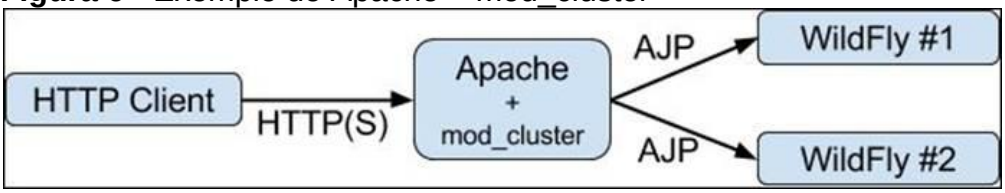
2.5. APACHE HTTP + MOD_CLUSTER

Segundo o portal CANALTECH, o servidor Apache ou Servidor HTTP Apache é o mais bem sucedido servidor web livre que existe bastante popular e utilizado principalmente no Linux, é responsável por disponibilizar páginas e todos os recursos que podem ser acessados pelo internauta. Um dos destaques do Apache HTTP é que apesar de tudo, ele é distribuído sob licença GNU, ou seja, é gratuito, podendo ser estudado e modificado através de seu código fonte por qualquer pessoa.

Sobre o mod_cluster do Apache, o Portal MASTERTHEBOSS (2010) diz:

> O mod cluster é um balanceador de carga baseado em http que simplifica muito a configuração de um cluster http. Ele é fornecido como um conjunto de módulos que precisam ser instalados no servidor httpd e uma Biblioteca de Arquivo de Serviços (.sar) que precisa ser implantada no JBoss AS (Ele é incorporado no JBoss AS 6/7).

Figura 6 - Exemplo do Apache + mod_cluster



Fonte: <http://apprize.info/javascript/wildfly_2/wildfly_2.files/image089.jpg> Acesso em: 15 de Setembro de 2017

2.6. JBOSS / Wildfly + MODO DOMAIN

Segundo o Portal JBUG-BRASIL, o Wildfly, antes conhecido como JBoss, é um servidor de aplicação Java EE desenvolvido em Java, que pode ser executado em qualquer Sistema Operacional, 32 ou 64 bits que tenha suporte ao Java.

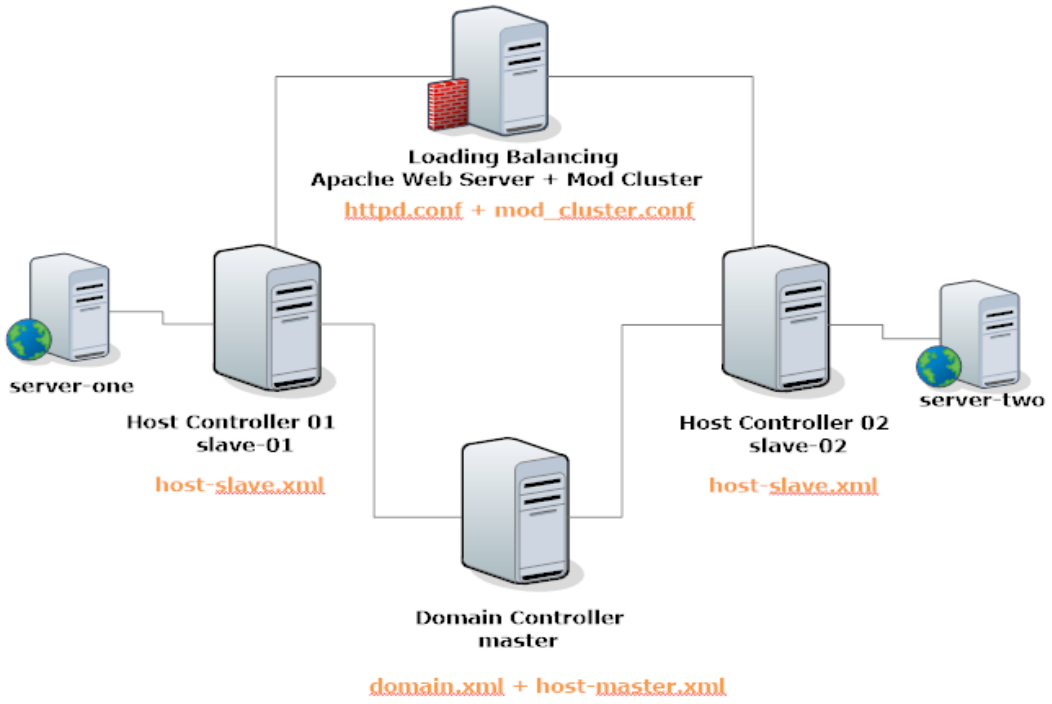
Sobre a mudança do nome de JBoss para Wildfly, o portal JBUG-BRASIL explica:

> O principal motivo da alteração ocorreu pelo fato da semelhança entre o *JBoss Application Server(AS)* e o *JBoss Enterprise Application Platform(EAP)* e também outros produtos do portifólio *JBoss* gerar um pouco de confusão entre eles.

Sobre o Modo Domínio do Wildfly, o Portal JBUG-BRASIL que:

> O modo Domain é o modo que foi introduzido no JBoss AS 7 onde é possível gerenciar um conjunto de instâncias Wildfly, agrupando-os e assim permitindo compartilhar configurações comuns entre eles. Além de compartilhar configurações, é possível também através de um único console de gerenciamento iniciar ou parar instâncias (ou grupos inteiros), verificar seu status e estatísticas de cada subsystem, etc.

Figura 7 - Exemplo de Ambiente com Apache + mod_cluster com o Modo Dominio do Wildfly



Fonte: <https://jbossdivers.files.wordpress.com/2012/12/mododomain.png> Acesso em: 15 de Setembro de 2017

2.7. TOMCAT

Conforme D'AVILA (2003), O Tomcat é um servidor de aplicações Java para Web, é livre e de código aberto que surgiu dentro do conceituado projeto Apache Jakarta, que teve o apoio e endosso oficial da Sun Microsystems como implementação de Referência (RI) para as tecnologias Java Servlet e Java Server Pages (JSP). Atualmente, o Tomcat possui seu próprio projeto de desenvolvimento independente, dentro da Apache Software Foundation.

Sobre utilizar o Tomcat também como Servidor Web, D'AVILA (2003) explica que, tecnicamente, o Tomcat é um Container Web, que é parte da corporativa Java EE (Java Enterprise Edition) que abrange as tecnologias Servlet e JSP, com isso ele tem a capacidade também de agir como servidor web/http autônomo.

2.8. POSTGRESQL

Segundo o site PGDOCPTBR, O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional, de código aberto, que suporta grande parte do padrão SQL e oferece várias funcionalidades, como:

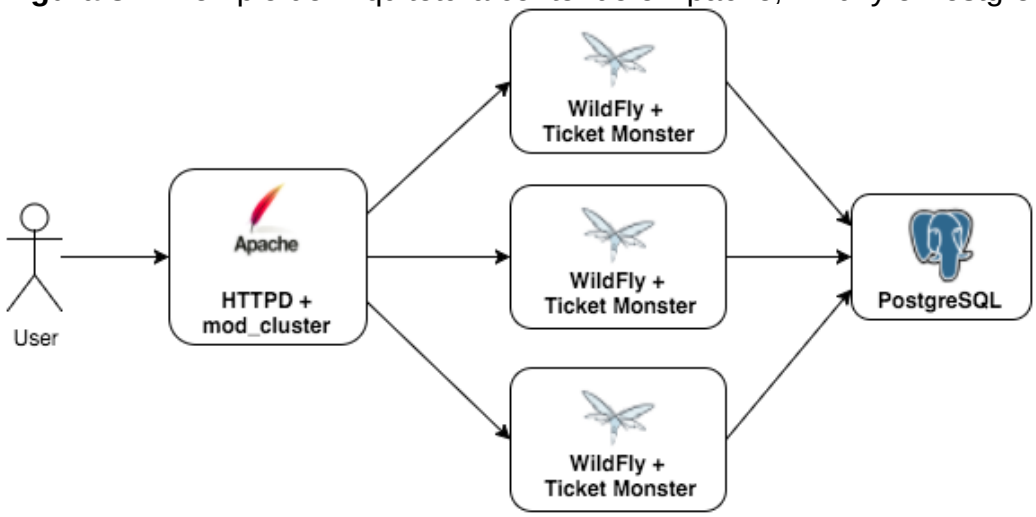
- Chave estrangeira;
- Gatilho;
- Visões;
- Integridade Transacional;
- Controle de simultaneidade multiversão;

Além disso, o PostgreSQL pode ser ampliado pelos usuários de várias maneiras, como por exemplo, adicionando:

- Tipos de dado;
- Funções;
- Operadores;
- Funções de agregação;
- Método de índice;
- Linguagens procedurais;

Por ser um projeto open source, o PostgreSQL pode ser utilizado, modificado e distribuído por qualquer pessoa para qualquer finalidade, seja particular, comercial ou acadêmica, livre de encargos.

Figura 8 - Exemplo de Arquitetura contendo o Apache, Wildfly e PostgreSQL



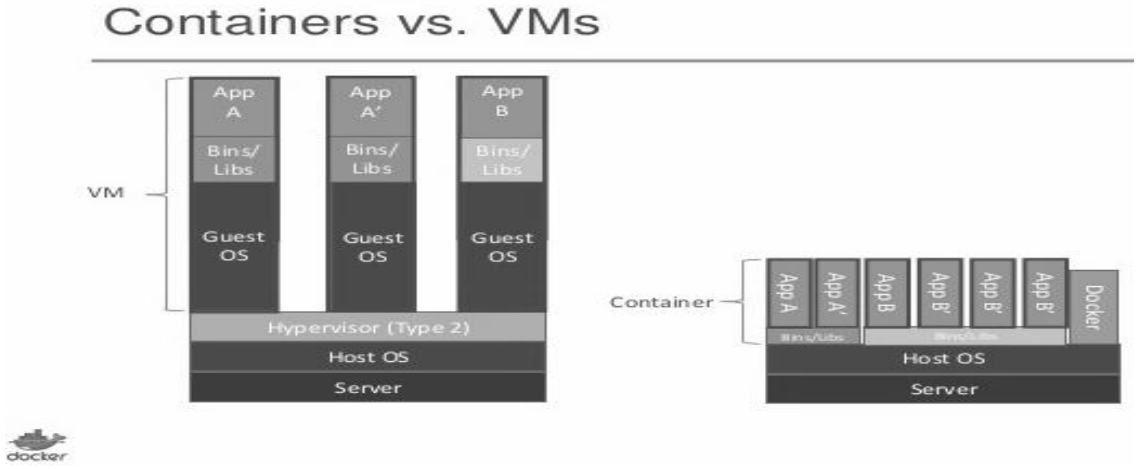
Fonte: <http://rafabene.com/images/docker_mod_cluster.png> Acesso em: 15 de Setembro de 2017

2.9. DOCKER

Segundo GOMES et al. (2015):

> O Docker é uma plataforma que automatiza a implantação de aplicações dentro de ambientes isolados denominados containers. Trata-se, portanto, de uma solução para desenvolvedores e administradores de sistema desenvolverem, embarcarem, integrarem e executarem aplicações rapidamente. Seu principal objetivo é proporcionar múltiplos ambientes isolados dentro do mesmo servidor, mas acessíveis externamente via tradução de portas.

Figura 9 - Containers x Vms



Fonte: <https://www.docker.com> Disponível em: 15 de Setembro de 2017

Ainda GOMES et al, (2015), sobre a figura acima que mostra a diferença entre Maquina Virtual e Container, eles explicam que:

> [...] a solução pode aparentar uma semelhança à estrutura de ambiente virtualizado, porém difere no que tange à necessidade de uma camada intermediária de sistema operacional entre o hospedeiro e as aplicações hospedadas, que no caso do Docker é desnecessária, pois ela utilizar o mesmo kernel, criando ambientes isolados a nível de disco, memória e processamento.

GOMES et al. (2015) explica ainda sobre um artefato do Docker, o Dockerfile:

> O Dockerfile, é o arquivo que contém todos os comandos que normalmente seriam usados para configurar manualmente a implantação do ambiente. A imagem será criada com base nos parâmetros especificados nesse arquivo.

Exemplo de arquivo Dockerfile.

```
FROM debian
MAINTAINER rafael.gomes@ufba.br
EXPOSE 8080
ENV TOMCAT_VERSION 7.0.62
ENV DEPLOY_DIR /maven
# Baixar e descompactar o Tomcat
RUN wget
http://archive.apache.org/dist/tomcat/tomcat-7/v\${TOMCAT_VERSION}/bin/
apache-tomcat-\${TOMCAT_VERSION}.tar.gz -O /tmp/catalina.tar.gz && tar xzf
/tmp/catalina.tar.gz -C /opt && ln -s /opt/apache-tomcat-
\${TOMCAT_VERSION}
/opt/tomcat && rm /tmp/catalina.tar.gz
ENV CATALINA_HOME /opt/tomcat
ADD deploy-and-run.sh /opt/tomcat/bin/deploy-and-run.sh
CMD /opt/tomcat/bin/deploy-and-run.sh
```

Para Finalizar GOMES et al. (2015) diz:

> Docker também estabelece um padrão de empacotamento de soluções. Ou seja, uma vez criada a estrutura do ambiente ela pode ser facilmente replicada, usada como referência para a criação de novas estruturas.

2.10. APACHE JMETER

Segundo o Manual da PAPPE (2013), o JMeter é uma ferramenta desktop para testes de performance, desenvolvida utilizando a linguagem Java, primeiramente utilizada para realizar testes em aplicações web, mas tem expandido suas funcionalidades, podendo realizar testes funcionais, em banco de dados entre outros.

O Manual da PAPPE (2013) mostra também os tipos de teste disponíveis:

> **Teste de Performance**: Os testes de performance tem como finalidade verificar o desempenho do sistema em condições normais de uso, onde o foco é obter informações relevantes da utilização das principais funções.
>
> **Teste de Carga**: Os testes de carga, diferentemente dos testes de performance, tem como objetivo a verificação do comportamento do produto com uma determinada quantidade de usuários.
>
> **Teste de Stress**: O objetivo de um teste de stress é explorar os limites do sistema, aumentando a carga indefinidamente até provocar um “crash”. A principal diferença entre o teste de carga e o teste performance é que, enquanto o primeiro preocupa-se em determinar como o sistema se comporta em caso de um grande numero de acessos, o segundo tem por objetivo testar a quantidade máxima de usuários, determinando, assim, Seu limite de utilização.

Um dos principais relatórios gerados pelo JMeter é o relatório de sumário, e as métricas, segundo PAPPE (2013) são as seguintes:

- **Rótulo** - Rótulo do elemento de requisição;
- **Amostras** - Quantidade de amostras, isto é, pedidos de requisição HTTP que ocorreu para o determinado segmento.
- **Média** - Tempo médio, em milissegundos, de resposta para determinado pedido de requisição HTTP.
- **Min** - Tempo mínimo, em milissegundos, de resposta para um determinado pedido de requisição HTTP.
- **Max** - Tempo Maximo, em milissegundos, de resposta para um determinado pedido de requisição HTTP.
- **Desvio Padrão** - O desvio padrão apresenta os casos em que determinadas amostras se distanciam do comportamento médio das demais amostras em razão do tempo de resposta. Quanto menor este valor mais consistente é o padrão de tempo das amostras coletadas.
- **% de Erro** - Porcentagem de erros nas amostras executadas.
- **Vazão** - É a medida da quantidade de requisições por unidade de tempo.
- **KB/sec** - Medida do “Throughput” em Kilobytes por segundo.
- **Média de Bytes** - Tamanho médio das respostas das amostras em bytes.

3. CENÁRIO ANTERIOR

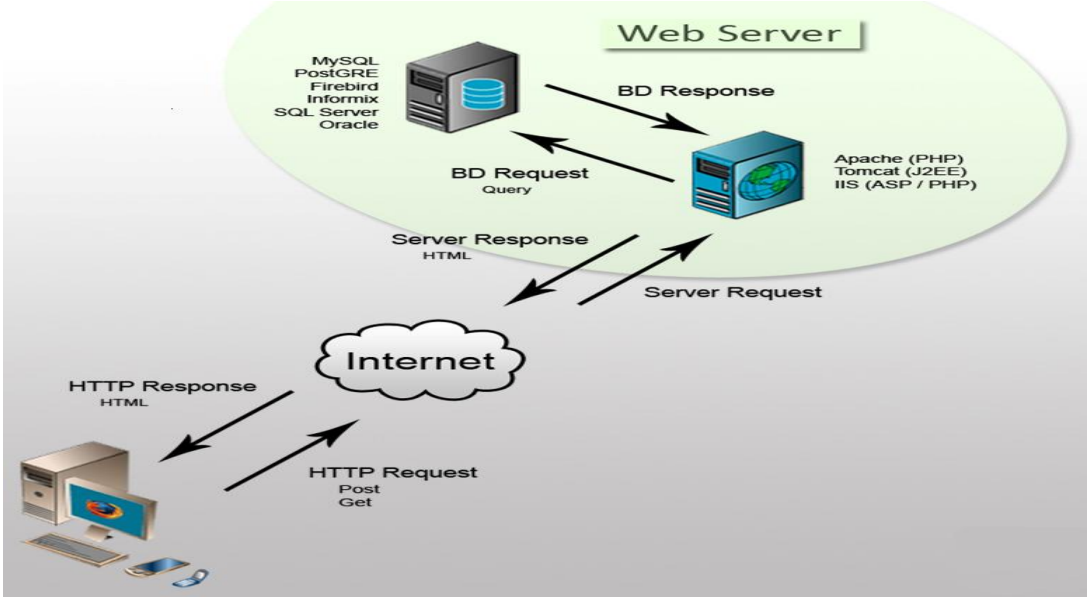
3.1. ARQUITETURA DE HARDWARE E SOFTWARE

- Processador: Intel I3 540 2 Núcleos 3.07 GHz;
- Memória RAM: 12 GB DDR 3 1333 MHz;
- Sistema Operacional: Windows 7 Professional;
- Armazenamento: HD SATA 7200rpm 500gb;
- Tomcat 7.0.72;
- PostgreSQL 9.5;

3.2. ARQUITETURA

O Ambiente que comporta a aplicação é bastante simples, utiliza o modelo arquitetural de 3/4 camadas, onde há o cliente (Navegador), a Camada de Apresentação e Lógica como o próprio Servidor de Aplicação, no caso o Tomcat 7.0.72, pois ele possui um servidor web embutido, e por ultimo a camada de Persistência que é o servidor de banco de dados, PostgreSQL 9.5, exemplificando essa estrutura na imagem a seguir;

Figura 10 - Funcionamento de uma aplicação Web no Modelo Arquitetural 4 Camadas



Fonte: <https://flavioaf.wordpress.com/2010/02/25/tutorial-entendendo-java-para-web-parte-1/>
Disponível em: 15 de Setembro de 2017

3.3. SEGURANÇA DA APLICAÇÃO

Foi verificado pelo log do servidor de aplicação Tomcat, que a aplicação estava sendo atacada constantemente pelo método chamado "Força bruta", possivelmente causada pela exposição direta do servidor de aplicação a internet, este método visa descobrir a senha via tentativa e erro, de todas as combinações possíveis ou pré definidas, como mostra a seguir:

```
mar 08, 2017 12:36:08 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "admin"
ADVERTÊNCIA: An attempt was made to authenticate the locked user "tomcat"
mar 08, 2017 12:36:39 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "manager"
mar 08, 2017 12:37:01 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "both"
mar 08, 2017 12:37:32 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
mar 08, 2017 12:38:05 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "a%09in"
mar 08, 2017 12:38:36 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "root"
mar 08, 2017 12:38:56 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "administrator"
mar 08, 2017 12:39:12 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
```

```
mar 08, 2017 12:39:12 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "inentadmin"
mar 08, 2017 12:39:41 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "pisces"
mar 08, 2017 12:40:41 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user
"chongchujianghu"
mar 08, 2017 12:41:13 PM org.apache.catalina.realm.LockOutRealm
filterLockedAccounts
ADVERTÊNCIA: An attempt was made to authenticate the locked user "user"
```

4. ESTUDO DE CASO - AMBIENTE PROPOSTO

Neste capítulo iremos fazer o passo a passo da implementação do ambiente proposto, toda ela foi baseada nas aulas da matéria de "Programação Concorrente e Distribuída" no curso de "Especialização em Desenvolvimento de Sistemas Corporativos" na instituição de ensino superior UNI-RN e em pesquisas na internet, citando como principais fontes os seguintes endereços:

- ✓ **Configurando Wildfly com Apache, load balance e clusterização no CentOS7 -** CARDOZO **;**
- ✓ **Load balancer com Wildfly 10 -** SCHIMIDT 2016;
- ✓ **Como Instalar o Docker no Ubuntu 16.04 -** DIGITAL OCEAN 2017;
- ✓ **Canal LinuxtTips -** FERNANDO;

Todos os comandos a seguir foram executados em uma Maquina Virtual com as seguintes configurações:

- Processador: 2 Núcleos;
- Memória RAM: 5392mb;
- Armazenamento: 32GB;
- Sistema Operacional: Ubuntu 16.04 LTS;
- Docker: Version 17.05.0-ce;

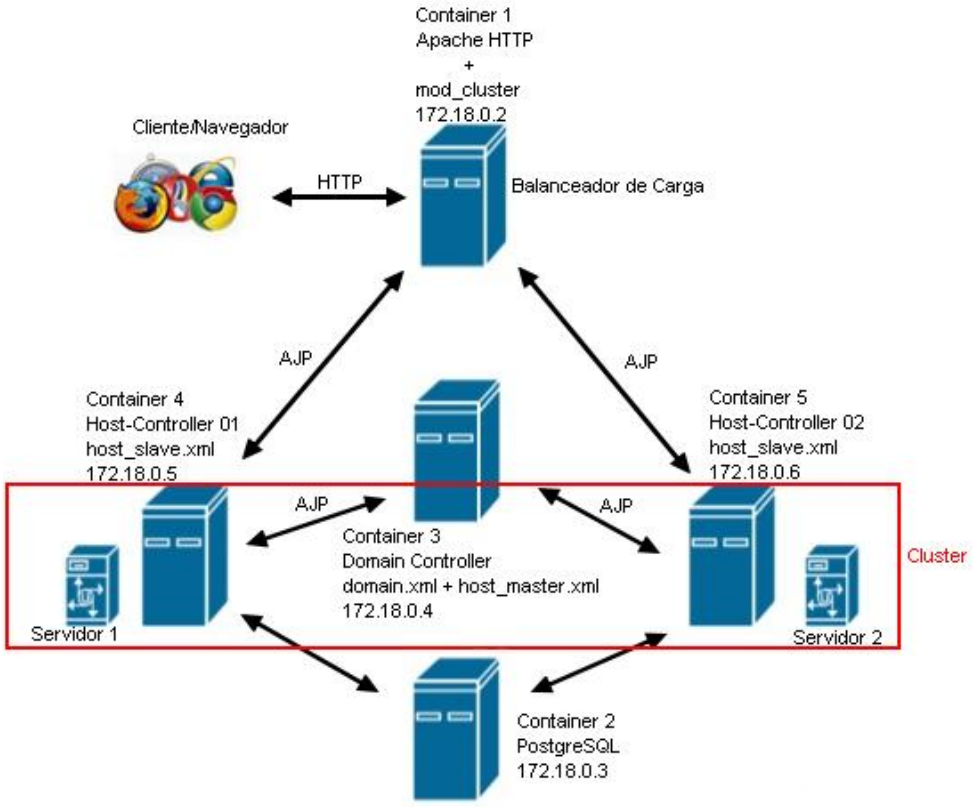
Para simplificar, os comandos necessários serão inseridos em caixas de texto, onde o que estiver em negrito deve ser executado no terminal.

4.1. ARQUITETURA

A arquitetura do ambiente proposto utiliza o modelo arquitetural multicamada, onde teremos cliente, que acessará o sistema por um navegador web, a camada de apresentação é o servidor Web (container 1), que funcionará como um balanceador de carga para acessar a camada de lógica que é formada por um cluster de servidores escravos do Wildfly (container 4 e 5), e a camada de persistência é o SGBD (container 2).

O Domain Controller (container 3) será a máquina responsável por receber e distribuir as configurações do Wildfly, como mostra a figura a seguir:

Figura 11 - Ambiente Proposto



Fonte: Próprio autor

4.2. INSTALAÇÃO DO DOCKER

Nos comandos a seguir, iremos fazer a instalação do Docker na máquina com Ubuntu, o Docker vai ser o diferencial desse ambiente, com ele vai ser possível criar toda a estrutura em uma única máquina distribuída em containers, estes comandos devem ser inseridos no terminal Linux.

```
$ sudo apt-get update
 $ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
 $ sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
 $ sudo apt-get update
 $ sudo apt-get install -y docker-engine
```

4.3. EXECUTANDO O COMANDO DOCKER SEM SUDO

Para facilitar o uso do Docker, devemos aplicar os comandos a seguir para não ser necessário utilizar o comando sudo toda vez que for executar um comando Docker (Será necessário finalizar a sessão ou reiniciar a máquina para o comando ter efeito).

```
$ sudo usermod -aG docker $(whoami)
 $ sudo usermod -aG docker SEU_USUARIO
```

4.4. CRIANDO OS CONTAINERS

Com o Docker instalado, será criado a partir de agora os 5 Containers:

1. Apache HTTP + mod_cluster (apache) - Servidor WEB;
2. PostgreSQL (postgres) - Sistema Gerenciador de Banco de Dados;
3. Wildfly Master (master) - Controlador de Domínio do Wildfly;
4. Wildfly Slave 1 (slave1) - Instância do Wildfly;
5. Wildfly Slave 2 (slave2) - Instância do Wildfly;

Para a criação de cada container, utilizaram-se imagens já prontas disponíveis no Docker hub, três criadas por mim, e uma distribuída pela própria plataforma, todas elas disponíveis nos seguintes endereços.

1. ApacheHTTP + mod_cluster - https://hub.docker.com/r/ederbrendo/apache/
2. PostgreSQL - https://hub.docker.com/_/postgres/
3. Wildfly Master - https://hub.docker.com/r/ederbrendo/wildfly-master/
4. Wildfly Slave - https://hub.docker.com/r/ederbrendo/wildfly-slave/

Os Dockerfiles das imagens criadas por mim estão disponíveis nos anexos:

1. Apache HTTP + mod_cluster - **ANEXO A**;
2. Wildfly Master - **ANEXO B**;
3. Wildfly Slave - **ANEXO C**;

Antes de criar os containers, deve-se criar uma rede própria do Docker, com esta rede, será atribuído um endereço fixo para cada container, para isso basta executar o seguinte comando:

```
$ docker network create --subnet 172.18.0.0/16 rede
```

4.4.1. CONTAINER 1 - APACHE

Para criar o container apache, que irá conter o Apache HTTPD + mod_cluster, devemos executar o comando a seguir no terminal:

```
$ docker run --net rede --ip 172.18.0.2 --restart=always --name apache -h apache --privileged -p 8080:8080 -d ederbrendo/apache
```

Entendendo o comando:

- **docker run** : execute;
- **--net rede --ip 172.18.0.2** : Atribua a rede "rede" ao container, com o ip 172.18.0.2;
- **--restart=always**: Execute o container após a reinicialização da máquina;

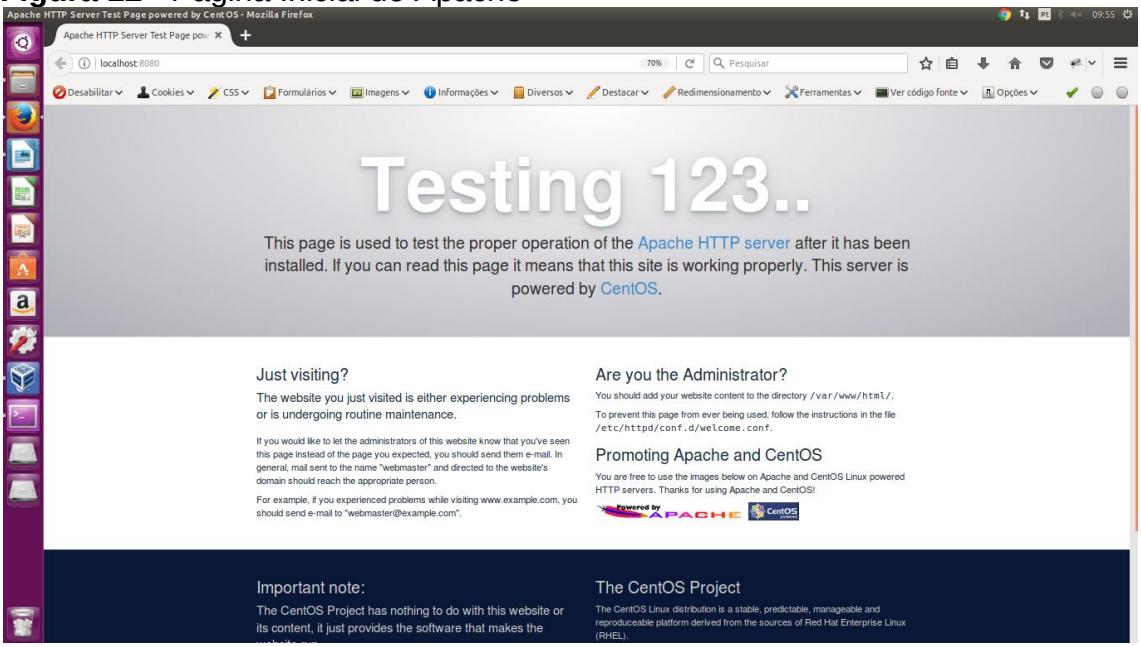
- **--name apache**: nome do container;
- **-- privileged** : Container com privilegios de administrador;
- **-p 8080:8080** : Atribuir a porta 8080 do host a 8080 do container;
- **ederbrendo/apache** : Nome da Imagem no Docker Hub;

Para testar se o container está ativo, executar o comando docker ps

```
$ docker ps
 CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
 41a1ef0645dc        ederbrendo/apache   "/sbin/init"        28 minutes ago
Up 28 minutes       0.0.0.0:8080->8080/tcp   apache
```

Para acessar a página inicial do Apache, basta acessar no navegador: http://localhost:8080

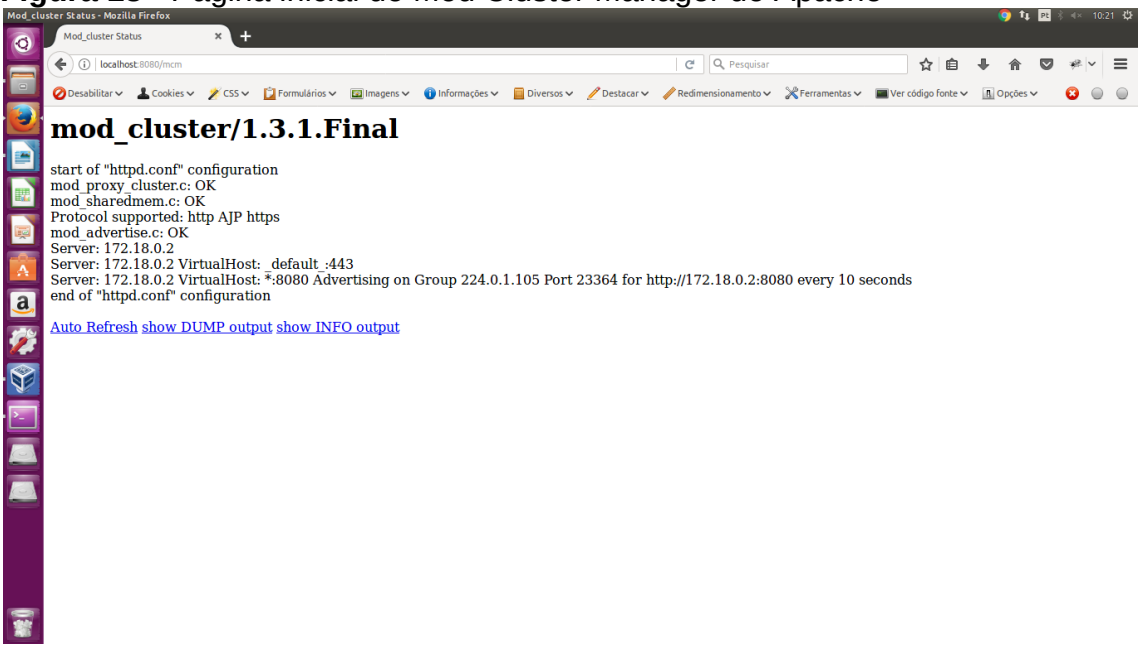
Figura 12 - Página Inicial do Apache



Fonte: Próprio autor

Para acessar a página do Mod Cluster Manager do Apache, basta acessar no Navegador: http://localhost:8080/mcm (User: admin, Password: admin).

Figura 13 - Página inicial do Mod Cluster Manager do Apache



Fonte: Próprio autor

4.4.2. CONTAINER 2 - POSTGRESQL

O container do PostgreSQL será criado pelo repositório oficial, portanto não necessitará de nenhuma configuração mais aprofundada, basta executar o seguinte comando no terminal:

```
$ docker run --net rede --ip 172.18.0.3 --restart=always --name postgresql -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres
```

Entendendo o comando:

- **docker run** : execute;
- **--net rede --ip 172.18.0.3** : Atribua a rede "rede" ao container, com o ip 172.18.0.3;
- **--restart=always**: Execute o container após a reinicialização da máquina;
- **--name postgresql**: nome do container;

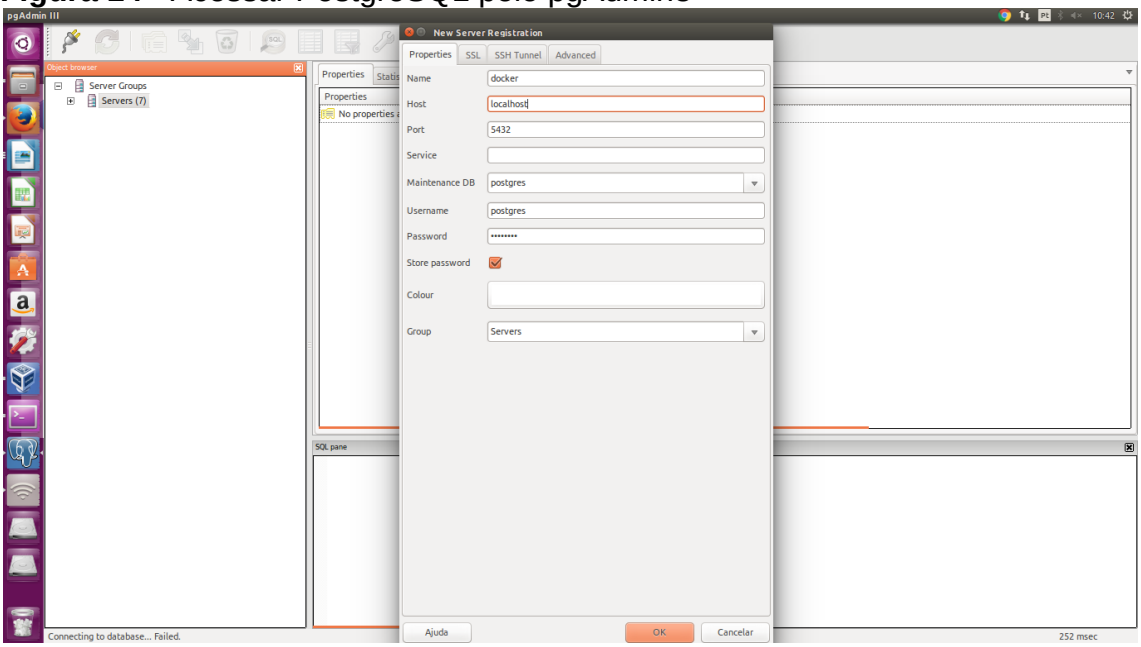
- **--POSTGRES_PASSWORD=postgres**: Senha do usuário postgres (Utilizei "postgres" como exemplo);
- **-- privileged** : Container com privilegios de administrador;
- **-p 5432:5432** : Atribuir a porta 5432 do host a 5432 do container;
- **postgres** : Nome da Imagem no Docker Hub;

Para verificar se o container está ativo, basta executar o comando docker ps.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
        STATUS              PORTS                    NAMES
fc82c1f5d0d9        postgres            "docker-entrypoint..."   4 seconds ago
        Up 3 seconds        0.0.0.0:5432->5432/tcp   postgresql
41a1ef0645dc        ederbrendo/apache   "/sbin/init"             About an hour ago
        Up About an hour    0.0.0.0:8080->8080/tcp   apache
```

Para se conectar pelo PGADMIN3, basta utilizar o seguinte endereço: http://localhost:5432

Figura 14 - Acessar PostgreSQL pelo pgAdmin3



Fonte: Próprio autor

4.4.3. CONTAINER 3 - WILDFLY MASTER

Para criar o container do Wildfly Master, basta executar o comando a seguir no terminal:

```
$ docker run --net rede --ip 172.18.0.4 --restart=always --name master -h master --privileged -p 9990:9990 -p 9999:9999 -d ederbrendo/wildfly-master
```

Entendendo o comando:

- **docker run** : execute;
- **--net rede --ip 172.18.0.4** : Atribua a rede "rede" ao container, com o ip 172.18.0.4;
- **--restart=always**: Execute o container após a reinicialização da máquina;
- **--name master**: Nome do container;
- **-- privileged** : Container com privilegios de administrador;
- **-p 9990:9990** : Atribuir a porta 9990 do host a 9990 do container (Manager do Wildfly);
- **-p 9999:9999** : Atribuir a porta 9999 do host a 9999 do container (Host do Wildfly);
- **ederbrendo/wildfly-master** : Nome da Imagem no Docker Hub;

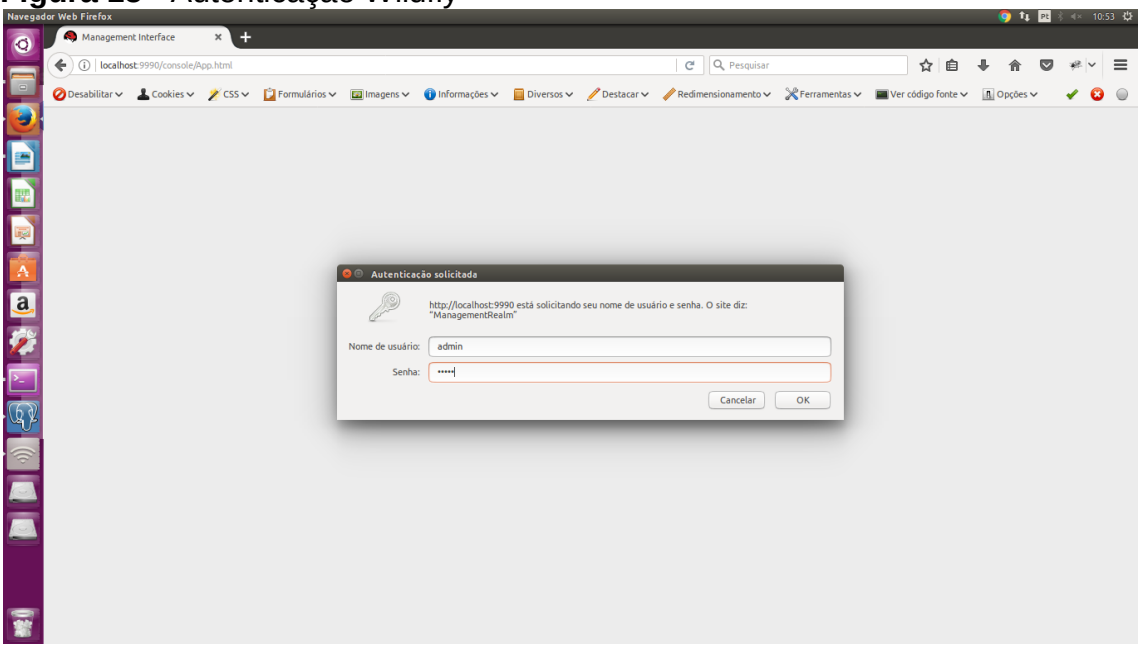
Para verificar se o container está ativo, basta executar o comando docker ps.

```
$ docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS              PORTS                                            NAMES
6b792ee62d99        ederbrendo/wildfly-master   "/sbin/init"             7 minutes ago
Up 7 minutes        0.0.0.0:9990->9990/tcp, 0.0.0.0:9999->9999/tcp   wildfly-master
fc82c1f5d0d9        postgres                 "docker-entrypoint..."   19 minutes ago
Up 19 minutes       0.0.0.0:5432->5432/tcp                           postgresql
41a1ef0645dc        ederbrendo/apache        "/sbin/init"             2 hours ago         Up
2 hours             0.0.0.0:8080->8080/tcp                           apache
```

Para acessar a pagina principal do Wildfly, basta utilizar o seguinte endereço no navegador: http://localhost:9990 (User: admin, Password: admin).

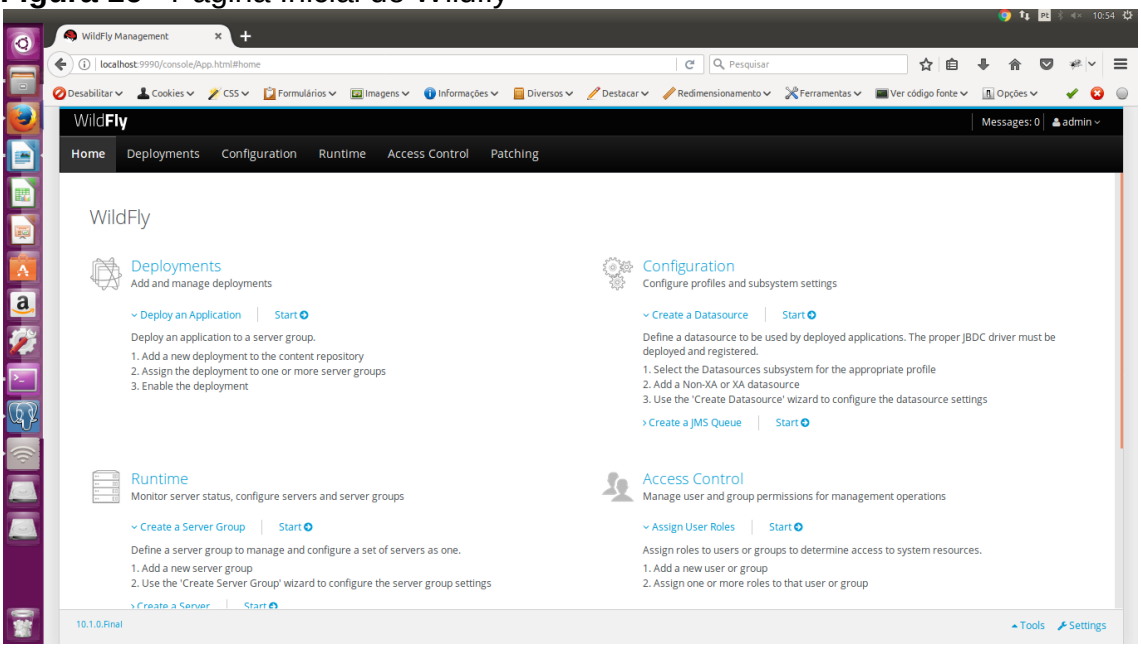
Para fazer o deploy da aplicação, basta se autenticar na página principal do Wildfly, ir em Deployments, Server Groups, arquitetura, Deployment, Add e selecionar seu arquivo .war.

Figura 15 - Autenticação Wildfly



Fonte: Próprio autor

Figura 16 - Página Inicial do Wildfly



Fonte: Próprio autor

4.4.4. CONTAINER 4 e 5 - WILDFLY SLAVE 1 e SLAVE 2

Iremos agora criar os dois últimos containers, estes serão os nós escravos (Cluster) do Wildfly, neste exemplo criaremos apenas dois, mas de acordo com o servidor, pode ser criado mais para uma maior disponibilidade da aplicação:

```
$ docker run --net rede --ip 172.18.0.5 --restart=always --name slave1 -h slave1 --privileged -d ederbrendo/wildfly-slave
$ docker run --net rede --ip 172.18.0.6 --restart=always --name slave2 -h slave2 --privileged -d ederbrendo/wildfly-slave
```

Entendendo o comando:

- **docker run** : Execute;
- **--net rede --ip 172.18.0.5 | --net rede --ip 172.18.0.6** : Atribua a rede "rede" ao container, com o ip 172.18.0.5 e 172.18.0.6;
- **--restart=always**: Execute o container após a reinicialização da máquina;
- **--name slave1 | slave2**: Nome do container;
- **-- privileged** : Container com privilegios de administrador;
- **ederbrendo/wildfly-slave** : Nome da imagem no Docker Hub;

Para verificar se os containers estão ativos, basta utilizar o comando docker ps:

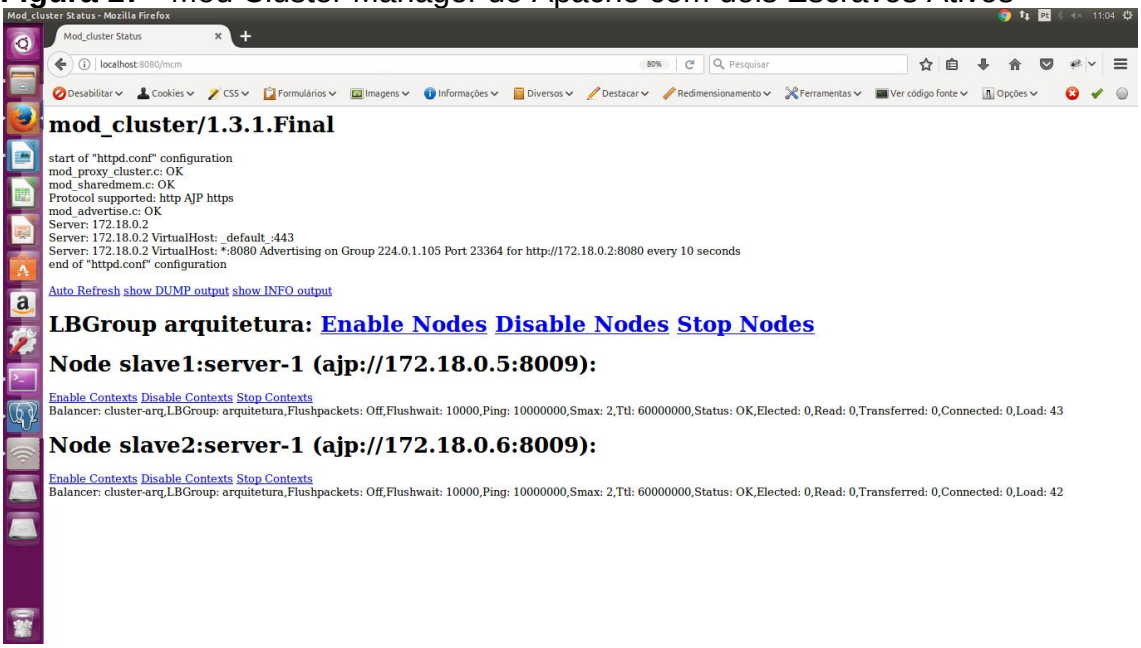
```
$ docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS              PORTS                                NAMES
b998e5f1725d        ederbrendo/wildfly-slave    "/sbin/init"             3 minutes ago
Up 3 minutes                                     slave2
159e11ba5398        ederbrendo/wildfly-slave    "/sbin/init"             3 minutes ago
Up 3 minutes                                     slave1
6b792ee62d99        ederbrendo/wildfly-master   "/sbin/init"             14 minutes ago
Up 14 minutes       0.0.0.0:9990->9990/tcp, 0.0.0.0:9999->9999/tcp   wildfly-
master
fc82c1f5d0d9        postgres                 "docker-entrypoint..."   26 minutes ago
Up 26 minutes       0.0.0.0:5432->5432/tcp               postgresql
41a1ef0645dc        ederbrendo/apache           "/sbin/init"             2 hours ago
Up 2 hours          0.0.0.0:8080->8080/tcp               apache
```

Para visualizar o log do console do Wildfly, basta executar o seguinte comando:

```
$ docker exec -ti NOME_CONTAINER more /var/log/wildfly/console.log
```

Para verificar se o ambiente está funcionando perfeitamente, basta acessar o Mod Cluster Manager do Apache e verificar se os Nós escravos foram criados.

Figura 17 - Mod Cluster Manager do Apache com dois Escravos Ativos



Fonte: Próprio autor

4.5. NAVEGANDO ENTRE OS CONTAINERS

Caso seja necessário fazer alguma alteração nos quatro containers criados por mim (apache, master, slave1 e slave2), é possível acessar eles via terminal, com o comando a seguir:

```
$ docker exec -ti NOME_CONTAINER /bin/bash;
```

Entendendo o comando:

- **docker exec -ti** : Execute e mantenha o terminal ativo;
- **NOME_CONTAINER** : Nome do container que será acessado;
- **/bin/bash**: Shell do CentOs;

Dentro do container, o uso é o mesmo de uma maquina linux qualquer via terminal, é como se estivesse conectado a maquina via ssh, para sair do container, basta executar no terminal:

- **$ exit**

5. RESULTADOS

Este capítulo mostrará testes comparativos entre dois ambientes, o ambiente denominado de Tomcat será o cenário anterior e o Docker será o ambiente proposto, seguindo todos os passos mostrados no capítulo anterior, as configurações da maquina de testes são:

- Notebook Dell 5537R;
- Processador: Intel I7 4500U 4 Núcleos 1.8 ~ 3.0 GHz;
- Memória RAM: 16 GB DDR3L 1600 MHz;
- Sistema Operacional: Windows 7 Professional;
- Armazenamento: HD SSD 256gb;
- Java 1.8.0;
- Apache JMeter 3.2;

Os testes de carga foram feitos com a ferramenta JMeter, a partir de uma rede externa e busca responder a seguinte questão:

- Qual ambiente possui maior disponibilidade?

Para avaliar os resultados, será utilizado o "Relatório de Sumário" do JMeter.

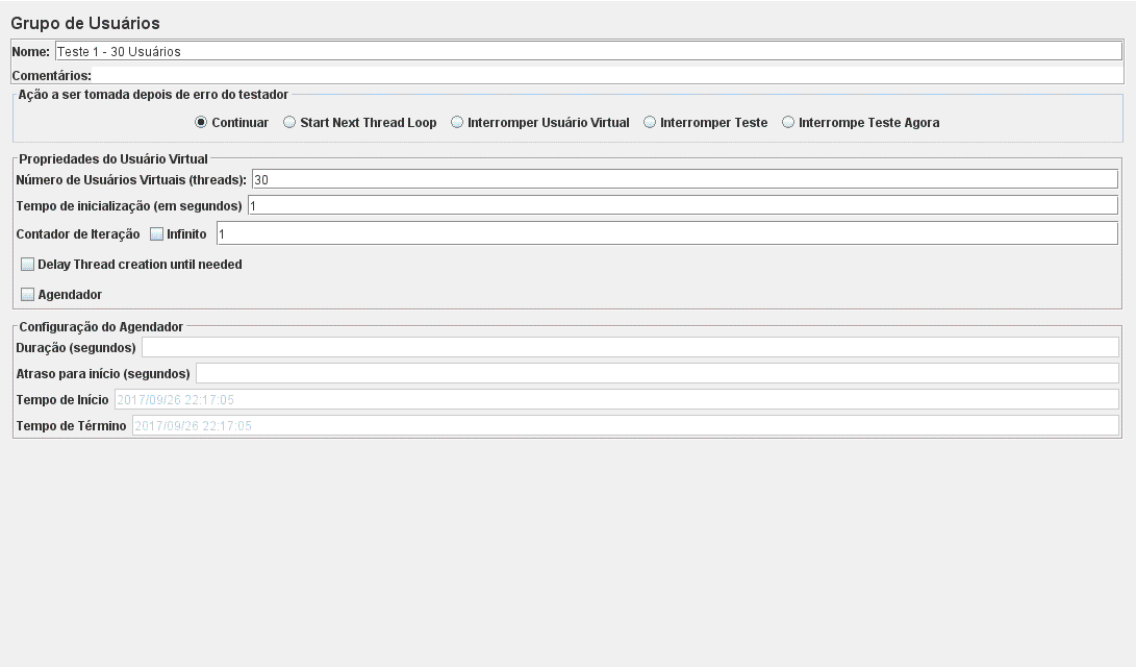
O Alvo do teste será uma página pública do sistema, onde serão feitas várias requisições HTTP do tipo GET de forma simultânea, com o resultado de cada teste, será comparado o tempo médio de resposta e a porcentagem de erros, foi definido um timeout de tempo de resposta de 30000ms, por tanto, caso o tempo de resposta seja maior que 30 segundos, o JMeter irá retornar erro de "SocketTimeoutException".

.

5.1. TESTE 01 - 30 USUÁRIOS

No Teste 01, 30 usuários virtuais irão realizar requisições HTTP simultaneamente, como mostra a imagem a seguir.

Figura 18 - Grupo de usuários



Fonte: Próprio autor

A Tabela 1 mostra que houve uma diferença na média de tempo de resposta entre os dois ambientes, enquanto o Tomcat teve como média 14782, o Docker teve 9891, portanto, o ambiente proposto obteve uma média de resposta de 33% melhor que o cenário anterior.

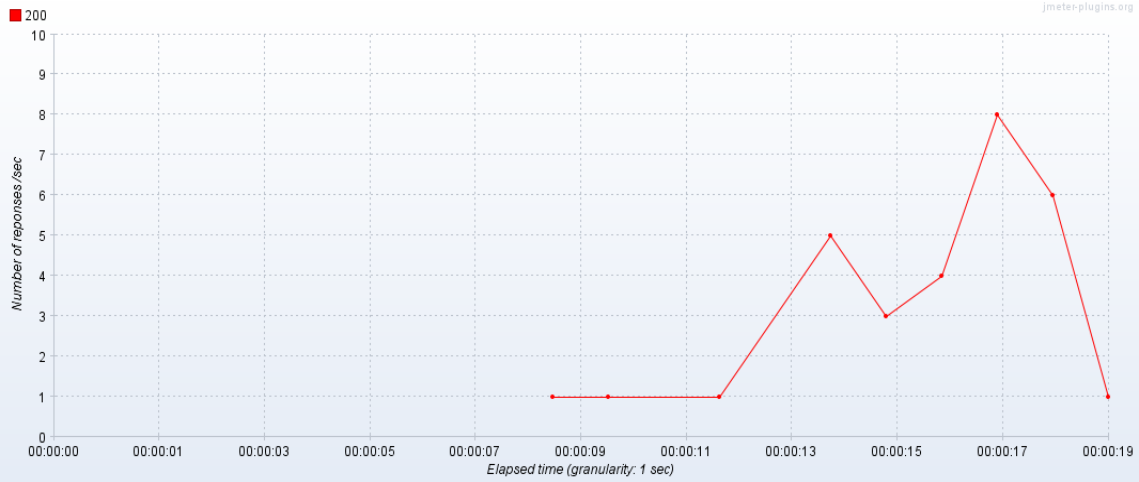
As figuras 21 e 22 mostram que os dois ambientes não retornaram erros.

Tabela 1 - Teste 1 - Relatório de Sumário

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Tomcat	30	14782	8479	17595	2256,98	0%	1,6/s	49,18	0,25	30837,0
Docker	30	9891	2568	15179	4239,17	0%	1,9/s	56,89	0,28	30926,0

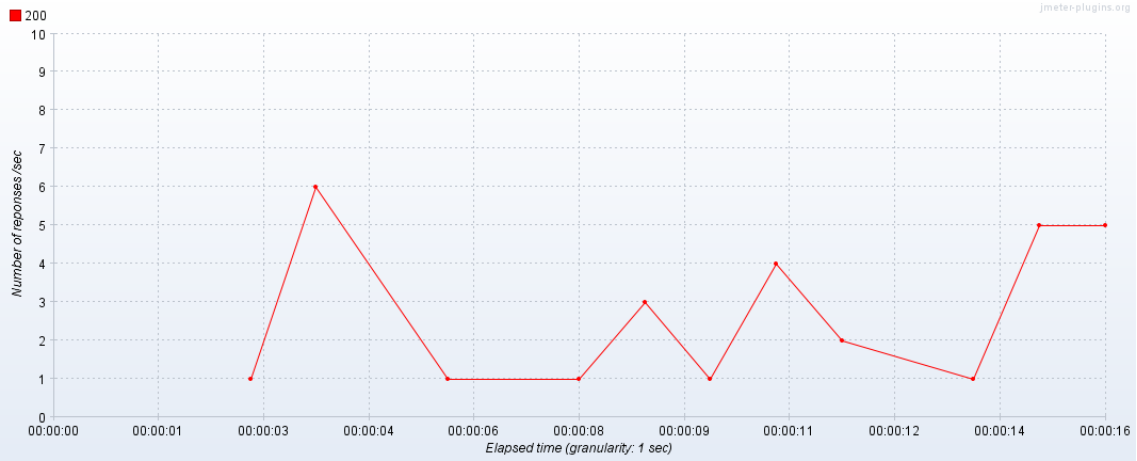
Fonte: Próprio autor

Figura 19 - Tomcat - Código de Respostas por Segundo



Fonte: Próprio autor

Figura 20 - Docker - Código de Repostas por Segundo

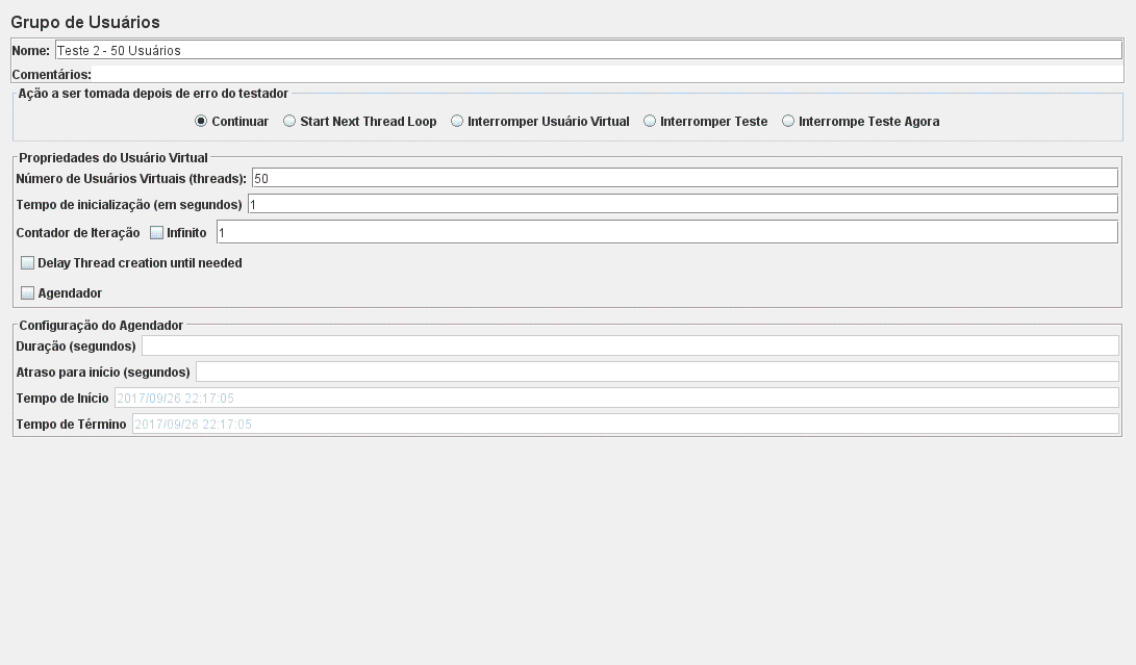


Fonte: Próprio autor

5.2. TESTE 02 - 50 USUÁRIOS

No Teste 02, 50 usuários virtuais irão realizar requisições HTTP simultaneamente, como mostra a imagem a seguir.

Figura 21 - Grupo de usuários



Fonte: Próprio autor

A Tabela 2 mostra que houve uma diferença na média de tempo de resposta entre os dois ambientes, enquanto o Tomcat teve como média 23360, o Docker teve 16915, portanto, o ambiente proposto obteve uma média de resposta de 27% melhor que o cenário anterior.

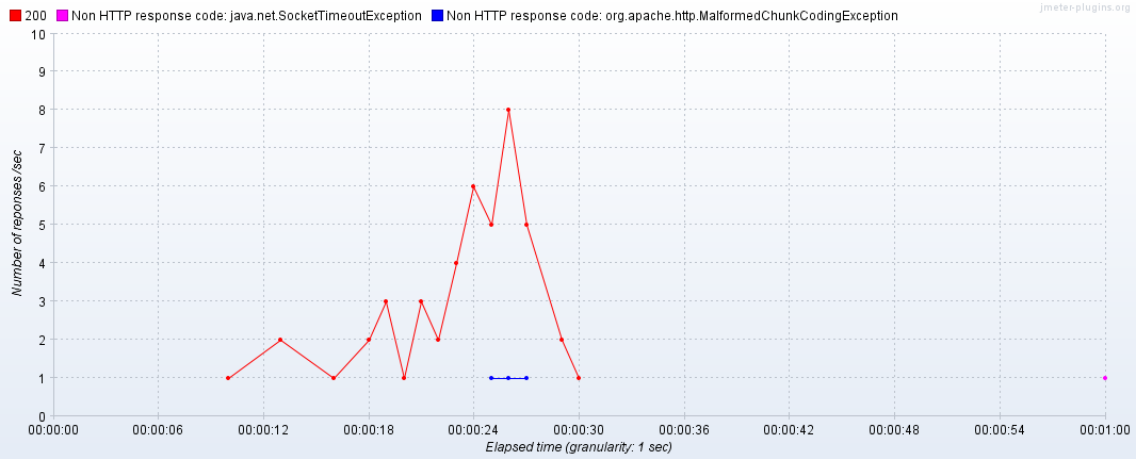
As figuras 22 e 23 mostram que o ambiente Tomcat retornou quatro erros, três de "MalformedChunkCodingException" e um de "SocketTimeoutException", enquanto o ambiente Docker não retornou nenhum.

Tabela 2 - Teste 2 - Relatório de Sumário

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Tomcat	50	23360	9248	59528	6616,03	8,0%	50,3/min	23,31	0,12	28481,70
Docker	50	16915	2792	25942	7820,66	0%	1,9/s	56,62	0,27	30926,0

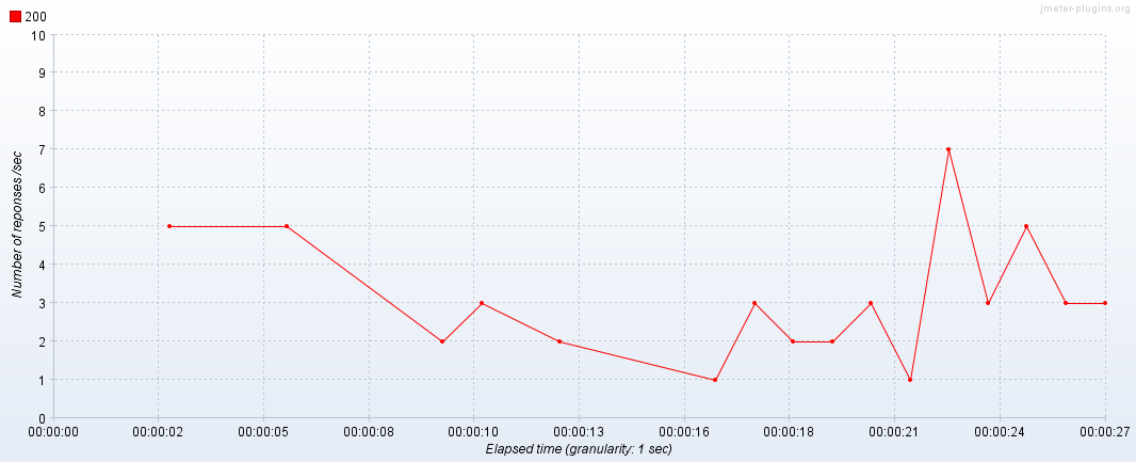
Fonte: Próprio autor

Figura 22 - Tomcat - Código de Respostas por Segundo



Fonte: Próprio autor

Figura 23 - Docker - Código de Repostas por Segundo

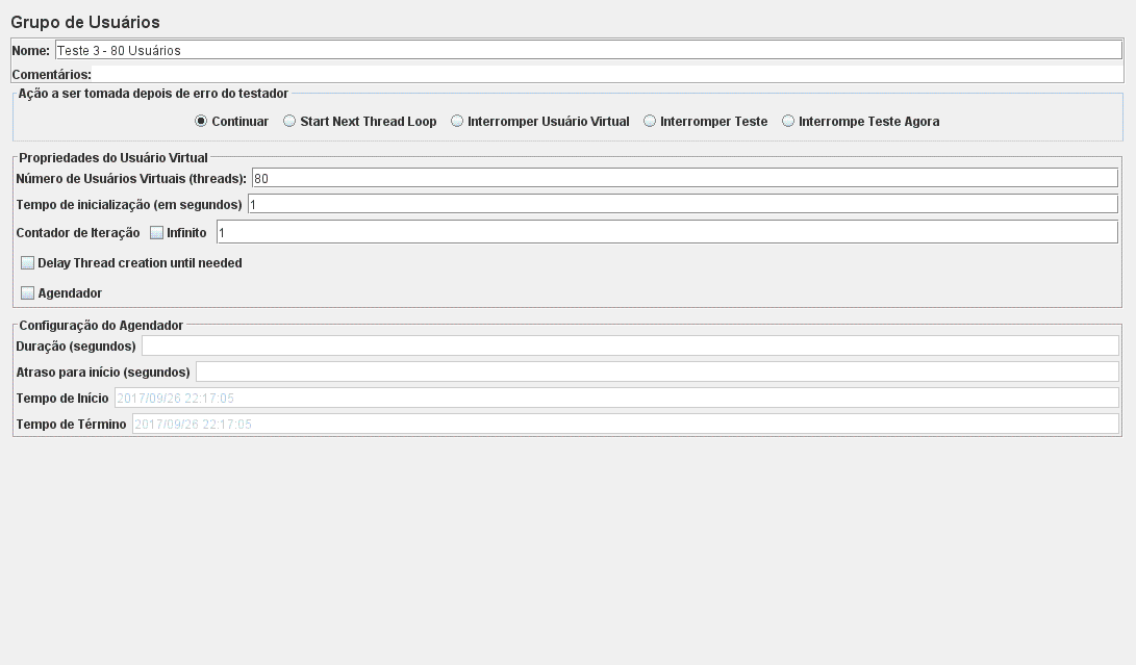


Fonte: Próprio autor

5.3. TESTE 03 - 80 USUÁRIOS

No Teste 03, 80 usuários virtuais irão realizar requisições HTTP simultaneamente, como mostra a imagem a seguir.

Figura 24 - Grupo de usuários



Fonte: Próprio autor

A Tabela 3 mostra que houve uma diferença na média de tempo de resposta entre os dois ambientes, enquanto o Tomcat teve como média 32449, o Docker teve 25955, portanto, o ambiente proposto obteve uma média de resposta de 20% melhor que o cenário anterior.

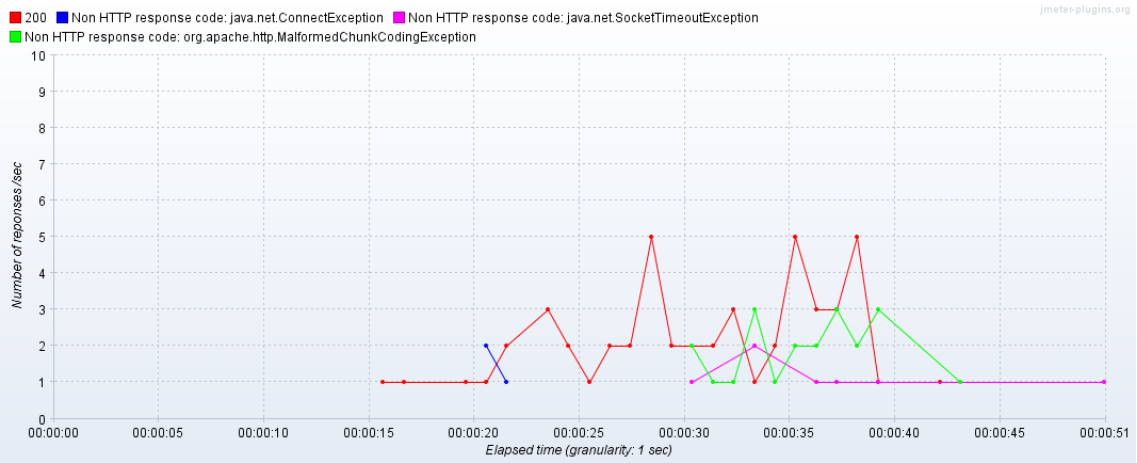
As figuras 22 e 23 mostram que os dois ambientes retornaram erros, O Tomcat retornou 31 erros, dois de "ConnectException", oito de "SocketTimeoutExcption" e 21 de "MalformedChunkCodingException", enquanto o Docker retornou seis erros, todos eles de "SocketTimeoutException".

Tabela 3 - Teste 2 - Relatório de Sumário

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Tomcat	80	32449	15851	50269	6616,72	38,75%	1,6/s	29,72	0,14	19471,2
Docker	80	25955	2782	45106	10921,93	7,50%	1,7/s	48,93	0,24	28807,8

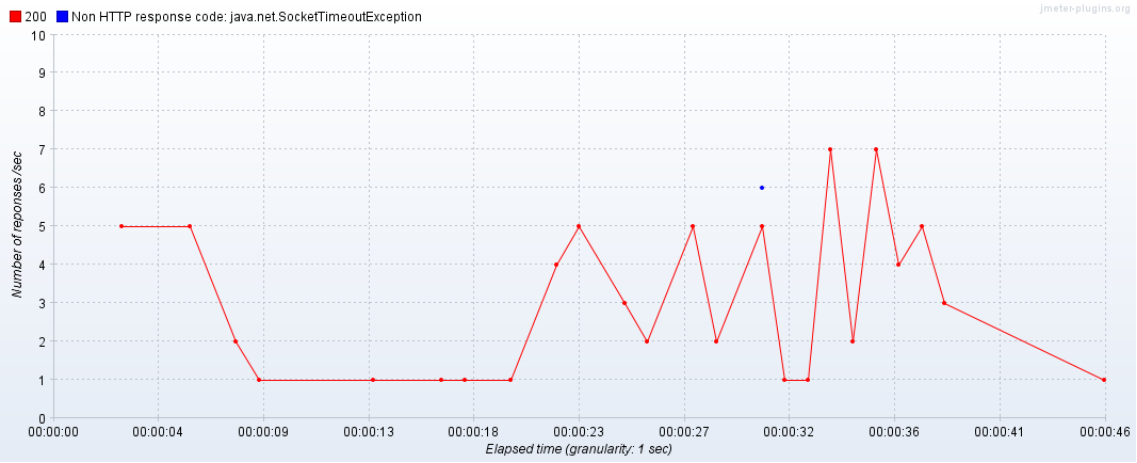
Fonte: Próprio autor

Figura 25 - Tomcat - Código de respostas por segundo



Fonte: Próprio autor

Figura 26 - Docker - Código de Repostas por Segundo



Fonte: Próprio autor

5.4. RESULTADO FINAL

Com base nos testes realizados, conclui-se que a aplicação foi mais disponível no ambiente proposto, das 160 amostras enviadas, a porcentagem de erro do ambiente proposto foi de 3,75% enquanto o cenário anterior retornou 21,88%, além disso, o tempo de resposta do ambiente proposto foi aproximadamente 23% mais rápido que o cenário anterior, como mostra a tabela 4.

Tabela 4 - Relatório Final

Rótulo	# Amostras	Média	% de Erro
Tomcat	160	26296	21,88%
Docker	160	20118	3,75%

Fonte: Próprio autor

6. CONCLUSÃO

O ambiente proposto foi implementado com sucesso, com o Docker, foi possível criar toda a estrutura em uma única máquina e sem a necessidade de manusear arquivos de configurações, que são duas das maiores dificuldades de se trabalhar com ambientes clusterizados para sistemas Java web.

Os resultados dos testes de carga mostraram que houve uma melhora significativa na disponibilidade da aplicação, onde somando os três testes, nas 160 amostras requisitadas, o cenário anterior retornou 21,88% de erros, enquanto o ambiente proposto retornou apenas 3,75%.

Outro benefício do novo ambiente é possuir duas instâncias do servidor de aplicação, onde caso algum falhe, o outro assumirá sem deixar a aplicação fora do ar, e ainda, se necessário, pode-se criar novas instâncias do servidor de aplicação, tornando o ambiente escalável.

Com a arquitetura multicamada, o problema de segurança do servidor de aplicação foi resolvido, onde o mesmo não é mais acessado diretamente pelo cliente.

Portanto, o ambiente proposto cumpriu com as expectativas, tornou a aplicação mais segura, disponível, escalável e sem custos de hardware.

7. REFERENCIAIS BIBLIOGRAFICOS

CANALTECH, **O que é Servidor Apache?**. Disponível em: <https://canaltech.com.br/internet/O-que-e-servidor-Apache/> Acesso em: 12 de Agosto de 2017.

CARDOSO, Claudio. **Configurando Wildfly com Apache, load balance e clusterização no CentOS7**. Disponível em: <http://www.ziben.com.br/configurando-wildfly-com-apache-load-balance-e-clusterizacao-no-centos7/> Acesso em: 18 de Setembro de 2017.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos, Conceitos e Projetos**. São Paulo: Bookman, 2008

D'AVILA, Márcio, **Tutorial Tomcat - Instalação e Configuração Básica**. Disponível em: <http://www.mhavila.com.br/topicos/java/tomcat.html> Acesso em: 18 de Agosto de 2017, 2003.

DIEDRICH, Cristiano, **O que é Docker?**. Disponível em: <https://www.mundodocker.com.br/o-que-e-docker/> Acesso em: 19 de Setembro de 2017

DIGITAL OCEAN. **Como Instalar e Usar o Docker no Ubuntu 16.04**. Disponível em: <https://www.digitalocean.com/community/tutorials/como-instalar-e-usar-o-docker-no-ubuntu-16-04-pt> Acesso em: 18 de Setembro de 2017.

FERNANDO, Jeferson. **Canal Linux Tips**. Disponível em: < https://www.youtube.com/user/linuxtipscanal> Acesso em: 18 de Setembro de 2017.

GOMES, Rafael; SOUZA, Rodrigo, **Docker - Infraestrutura como código, com autonomia e replicabilidade** , In: Superintendência de Tecnologia da Informação – Universidade Federal da Bahia (UFBA), Bahia, 2015.

JBUG-BRASIL, **WildFly – Desvendando do Básico ao Avançado**. Disponível em: <https://jboss-books.gitbooks.io/wildfly/> Acesso em: 12 de Agosto de 2017.

MACÊDO, Diego, **Arquitetura de Aplicações em 2, 3, 4 ou N camadas**, Disponível em: < http://www.diegomacedo.com.br/arquitetura-de-aplicacoes-em-2-3-4-ou-n-camadas/> Acesso em: 06 de Setembro de 2017.

MASTERTHEBOSS, **Clustering with Jboss mod_cluster**. Disponível em: <http://www.mastertheboss.com/jboss-server/jboss-cluster/clustering-with-jboss-modcluster> Acesso em: 12 de Agosto de 2017.

OPSERVICES, **Cluster: Definições e Caracteristicas**. Disponível em: <https://www.opservices.com.br/o-que-e-e-como-funciona-um-cluster/> Acesso em: 14 de Agosto de 2017.

PAPPE, Integração. **Manual de Utilização da Ferramenta JMeter**. Goiânia,2013. 29p. Manual Técnico. Instituto de Informática, Universidade Federal de Goiás.

PITANGA, Marcos. **Computação em Cluster**. Disponível em: <http://www.clubedohardware.com.br/artigos/redes/computa%C3%A7%C3%A3o-em-cluster-r33711/> Acesso em: 12 de Agosto de 2017.

PGDOCPTBR , **O Que é o PostgreSQL?**. Disponível em: <http://pgdocptbr.sourceforge.net/pg82/intro-whatis.html> Acesso em: 14 de Agosto de 2017.

SCHMIDT, Adriano. **Load balancer com Wildfly 10**. Disponível em: <http://localhost8080.blogspot.com.br/2016/09/load-balancer-com-wildfly-10.html> Acesso em: 18 de Setembro de 2017.

SILVA, Kleder, **Implementação de uma Infraestrutura de Cluster Virtualizada com Alta Disponibilidade,** In: Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Lins Prof. Antonio Seabra, Lins, 2012. 65p.

TANENBAUM, Andrew S., STEEN, Maarten Van. **Sistemas Distribuídos – Conceitos e Paradigmas.** 2. ed. São Paulo: Person, 2007.

8. ANEXOS

ANEXO A - Dockerfile Apache

```
FROM centos
MAINTAINER "Eder Brendo" <eder.brendo@gmail.com>
RUN yum update -y
RUN yum install openssh net-tools wget vim ntsysv nano -
y
RUN yum install httpd httpd-devel apr-devel openssl-
devel mod_ssl -y
RUN wget
http://downloads.jboss.org/mod_cluster//1.3.1.Final/linu
x-x86_64/mod_cluster-1.3.1.Final-linux2-x64-so.tar.gz
RUN tar zxvf mod_cluster-1.3.1.Final-linux2-x64-
so.tar.gz
RUN mv *.so /etc/httpd/modules

COPY mod_cluster.conf /etc/httpd/conf.d/mod_cluster.conf

RUN sed -i -e 's/LoadModule
proxy_balancer_module/#LoadModule
proxy_balancer_module/g' /etc/httpd/conf.modules.d/00-
proxy.conf

RUN htpasswd -b -c /etc/mcmpassword admin admin

RUN systemctl enable httpd.service

ENTRYPOINT ["/sbin/init"]
```

Fonte: Próprio autor

ANEXO B - Dockerfile Wildfly Master

```
FROM centos
MAINTAINER "Eder Brendo" <eder.brendo@gmail.com>

RUN yum update -y
RUN yum install openssh net-tools wget vim ntsysv nano
initscripts -y
RUN wget --no-check-certificate --no-cookies --header
"Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u131-
b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.rpm
RUN rpm -Uvh jdk-8u131-linux-x64.rpm
RUN alternatives --install /usr/bin/java java
/usr/java/latest/jre/bin/java 200000
RUN alternatives --install /usr/bin/javaws javaws
/usr/java/latest/jre/bin/javaws 200000
RUN alternatives --install /usr/bin/javac javac
/usr/java/latest/bin/javac 200000
RUN alternatives --install /usr/bin/jar jar
/usr/java/latest/bin/jar 200000
RUN wget
http://download.jboss.org/wildfly/10.1.0.Final/wildfly-
10.1.0.Final.tar.gz
RUN tar zxvf wildfly-10.1.0.Final.tar.gz
RUN mv wildfly-10.1.0.Final wildfly
RUN mv wildfly /opt
RUN groupadd wildfly
RUN adduser -g wildfly wildfly
RUN chown wildfly.wildfly -R /opt/wildfly*
RUN cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly.conf
/etc/default/
RUN cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly-
init-redhat.sh /etc/init.d/wildfly

COPY wildfly.conf /etc/default/wildfly.conf
COPY domain.xml /opt/wildfly/domain/configuration/domain.xml

RUN mkdir -p /opt/wildfly/modules/org/postgres/main

COPY module.xml
/opt/wildfly/modules/org/postgres/main/module.xml
COPY postgresql-9.4-1201-jdbc41.jar
/opt/wildfly/modules/org/postgres/main/postgresql-9.4-1201-
jdbc41.jar

RUN /opt/wildfly/bin/add-user.sh admin admin

RUN systemctl enable wildfly.service

ENTRYPOINT ["/sbin/init"]
```

Fonte: Próprio autor

ANEXO C - Dockerfile Wildfly Slave

```
FROM centos
MAINTAINER "Eder Brendo" <eder.brendo@gmail.com>

RUN yum update -y
RUN yum install openssh net-tools wget vim ntsysv nano
initscripts -y
RUN wget --no-check-certificate --no-cookies --header
"Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u131-
b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.rpm
RUN rpm -Uvh jdk-8u131-linux-x64.rpm
RUN alternatives --install /usr/bin/java java
/usr/java/latest/jre/bin/java 200000
RUN alternatives --install /usr/bin/javaws javaws
/usr/java/latest/jre/bin/javaws 200000
RUN alternatives --install /usr/bin/javac javac
/usr/java/latest/bin/javac 200000
RUN alternatives --install /usr/bin/jar jar
/usr/java/latest/bin/jar 200000
RUN wget
http://download.jboss.org/wildfly/10.1.0.Final/wildfly-
10.1.0.Final.tar.gz
RUN tar zxvf wildfly-10.1.0.Final.tar.gz
RUN mv wildfly-10.1.0.Final wildfly
RUN mv wildfly /opt
RUN groupadd wildfly
RUN adduser -g wildfly wildfly
RUN chown wildfly.wildfly -R /opt/wildfly*
RUN cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly.conf
/etc/default/
RUN cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly-
init-redhat.sh /etc/init.d/wildfly

COPY wildfly.conf /etc/default/wildfly.conf
COPY host-slave.xml /opt/wildfly/domain/configuration/host-
slave.xml

RUN mkdir -p /opt/wildfly/modules/org/postgres/main

COPY module.xml
/opt/wildfly/modules/org/postgres/main/module.xml
COPY postgresql-9.4-1201-jdbc41.jar
/opt/wildfly/modules/org/postgres/main/postgresql-9.4-1201-
jdbc41.jar

RUN systemctl enable wildfly.service


ENTRYPOINT ["/sbin/init"]
```

Fonte: Próprio autor