MINISTÉRIO DE MINAS E ENERGIA
Secretaria Executiva
Subsecretaria de Tecnologia e Inovação
Coordenação Geral de Inovação e Transformação Digital

Visão Geral Arquitetura JAVA

Versão 3.1

Histórico de Revisões

Data	Versão do documento	Descrição	Autor
06/03/2018	2.0	Atualizado com a definição da nova arquitetura.	Eric Azevedo - Tecnisys
29/03/2018	2.1	Retirada do WSO2, Retirada do Flyway e inclusão do KLB.	Eric Azevedo - Tecnisys
03/09/2018	2.2	Inclusão do item 3. Retirada do barramento de serviços.	Eric Azevedo - Tecnisys
04/01/2019	2.3	Alteração do item 3.	Eric Azevedo - Tecnisys
29/10/2019	2.4	Inclusão do item 4.	Eric Azevedo - Tecnisys
27/11/2022	2.5	Atualização do documento.	Marcos Martins - Tecnisys
05/04/2024	3.0	Atualização do documento	Luiz Alexandre
15/04/2024	3.1	Revisão	Carlos Gomes

Sumário

1. INTRODUÇÃO 4
1.1. Objetivo 4
1.2. Evolução deste documento 4
2. REPRESENTAÇÃO DA ARQUITETURA 4
2.1. Visão Geral 4
2.1.1. Front-End 4
2.1.2. Camada de Integração 6
2.1.3. Back-End 7
2.1.4. Infraestrutura 9
3. Estrutura do Docker e YML para as aplicações do MME 12
4. Informações para as contratadas no uso da arquitetura do MME 17

1. INTRODUÇÃO

1.1. Objetivo

Este documento tem como objetivo apresentar uma visão geral da arquitetura de sistemas Java utilizada no MME. Ela deve ser usada como referência para a construção de novos sistemas, ser fonte de consulta em relação ao padrão de arquitetura utilizado no MME e servir como insumo para a elaboração do Documento de Arquitetura do Projeto.

1.2. Evolução deste documento

Este documento pode ser continuamente adequado e adaptado conforme as lições aprendidas e as evoluções tecnológicas que surgirem no mercado.

2. REPRESENTAÇÃO DA ARQUITETURA

2.1. Visão Geral

No Front-End será indicado o Framework Angular e no Back-End, aplicações que irão disponibilizar serviços Rest com Spring Boot.

2.1.1. Front-End

O Front-End (camada de apresentação) é responsável pela lógica de apresentação dos dados e interação com usuário consumindo os serviços expostos pelo Back-End. Abaixo as tecnologias usadas:

- Angular:
    - Endereço: https://angular.io/guide/quickstart
    - Framework utilizado para desenvolver o Front-End da aplicação.

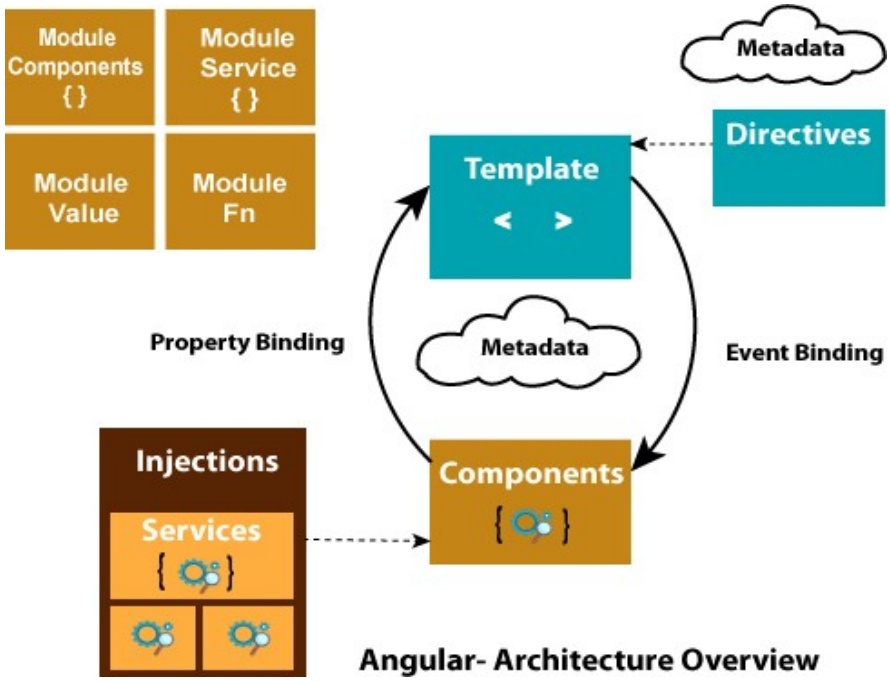


Fig.1 – Arquitetura Angular

- NodeJS:
    - Endereço: https://nodejs.org/en/download
    - Responsável pela instalação do Angular e utilização de suas dependência
    -

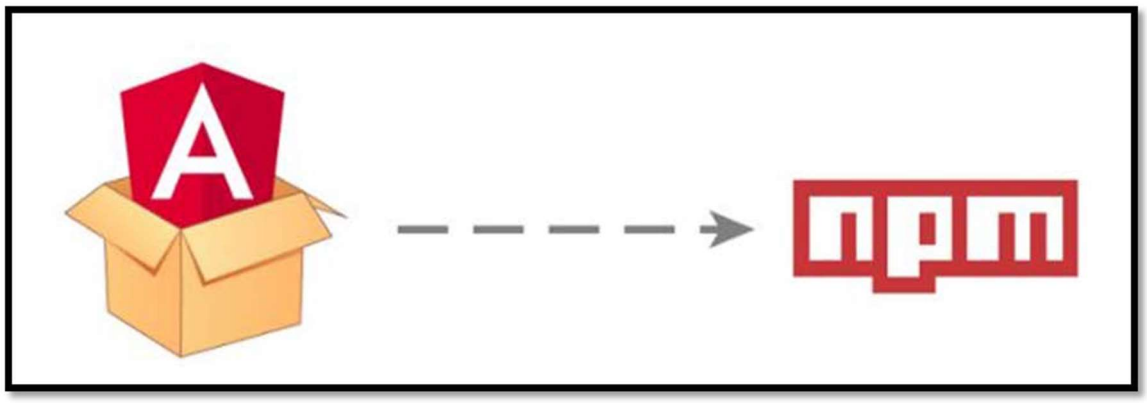


Fig.2 – O Angular ira utilizar o Node JS (npm) para buscar suas dependências, na criação do build do projeto.

2.1.2. Camada de Integração

Como dito anteriormente, para ligar o front-end e o back-end será utilizado um barramento de serviços, este barramento fará a ligação entre os dois “lados”, informado quais os serviços disponíveis e seus endereços.

2.1.3. Back-End

O Back-End é responsável por expor os serviços que serão consumidos pela aplicação, nestes serviços podemos ter desde acesso a base de dados, com ações de consultas, inserção ou alteração, até mesmo definições de regras de negócios pertinentes a aplicação. Abaixo as tecnologias usadas:

- Spring Boot:
  - Endereço: https://start.spring.io/
  - Tecnologia usada para expor e criar toda a estrutura dos Web-Services da aplicação.

- Spring Boot (actuator):
  - Actuator permite ao desenvolvedor monitorar e gerenciar aplicações, sem a necessidade de codificação e configuração. Essas informações de monitoramento e gerenciamento são expostas via REST.

- Spring Boot (jpa):
  - Para persistências e consultas ao banco de dados.

- Spring Boot (security):
  - Possibilita a filtragem de acessos, através de usuário e roles a um determinado item do web service e da aplicação.

□ Spring Boot (thymeleaf):

- Trata-se de um template engine para Java e uma alternativa ao JSP. Os templates são escritos, em sua maioria, com código HTML5 e tem boa integração com o Spring Framework. (Este componente deve ser usado para ajudar o usuário no momento de administrar tabelas de domínio no banco de dados).

□ Spring Boot (test):

- Necessário para a criação e execução dos testes unitários.

□ spring-test-dbunit:

- Simula os testes com dados de banco em memória, sem a necessidade de apagar ou editar um dado que esteja no banco fisicamente.

□ Spring Boot (undertow):

- Este componente empacota o projeto back-end no formato jar e encapsula junto a um servidor de aplicação. Aconselha-se usá-lo apenas durante o processo de desenvolvimento local, pois agiliza muito o projeto para os desenvolvedores. Porém ao envia-lo para o ambiente de desenvolvimento, homologação e produção do MME deve-se utilizar o formato war e subi-lo utilizando um servidor de aplicação como o Wildfly 10 ou superior.

2.1.4. Infraestrutura

O servidor de aplicação deverá ser usado o Wildlfly 10 ou superior, o mesmo poderá ser usado dentro de uma estrutura docker.

- Docker CE:
  - Endereço: https://docs.docker.com/install/
  - Ferramenta responsável por criar os containers e executar os servidores de aplicação. Aconselha-se utilizá-la apenas durante a fase desenvolvimento local, até o amadurecimento da estrutura do MME para o seu recebimento.

- Openshift:
  - Endereço: https://www.redhat.com/pt-br/technologies/cloud-computing/openshift
  - Openshift é um software que possibilita a criação de uma plataforma privada para criação de containers. Por meio dele, é possível administrar todos os pontos de seu ambiente Docker, incluindo: orquestração, load balance, volumes e rede. E ainda é possível fazer a diferenciação entre usuários, ou seja, você pode criar um pool de recursos para seu desenvolvedor.

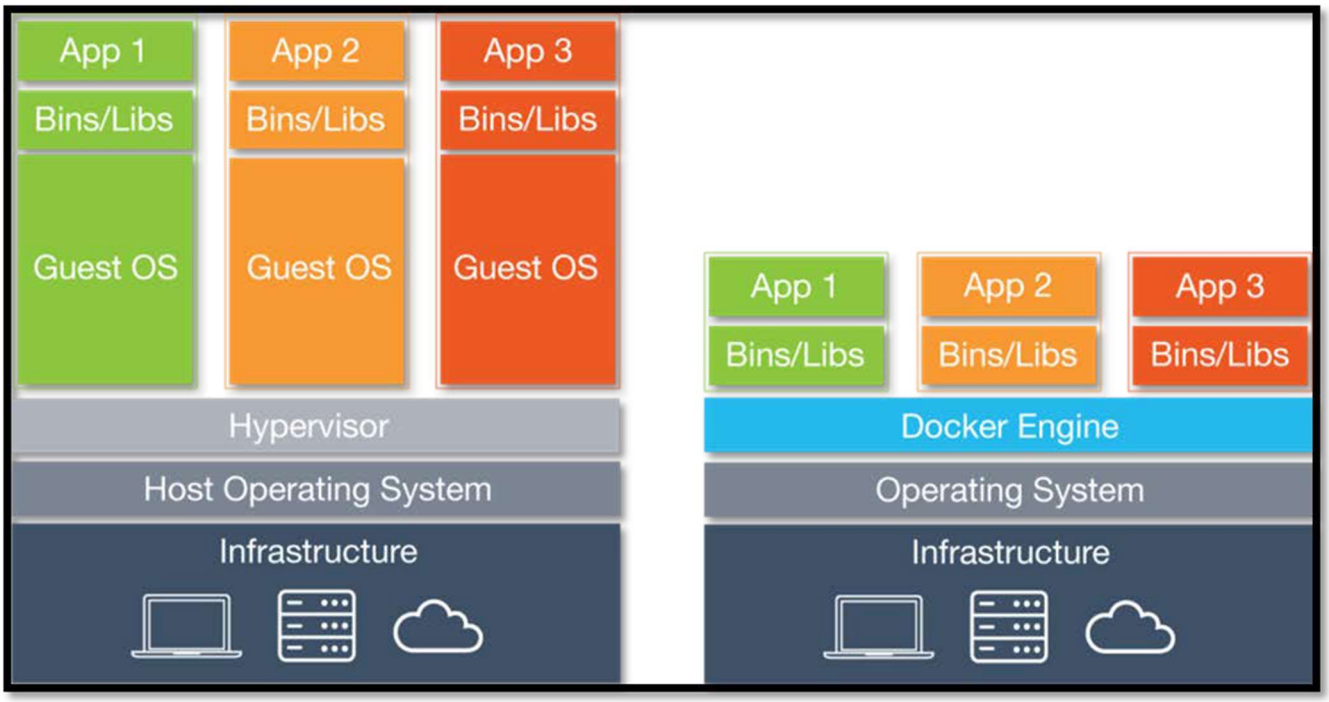


Fig.3 - Comparativo entre a arquitetura que utiliza como base máquinas virtuais e o Docker.

- Wildlfy:
    - Endereço: http://wildfly.org/downloads/
    - Servidor de aplicação que irá receber o front-end e o back-end da aplicação.

- Elastic Search:
    - Endereço: https://www.elastic.co/products/elasticsearch
    - O Elasticsearch é uma ferramenta de indexação textual altamente difundida.

- Logstash:
    - Endereço: https://www.elastic.co/products/logstash
    - Logstash facilita a leitura de logs (em um sentido amplo do termo, que inclui qualquer tipo de dado que represente eventos e seja gerado de forma automática, em um formato bem definido e continuamente).

- Kibana:
    - Endereço: https://www.elastic.co/products/kibana
    - Kibana permite a criação de gráficos a partir de dados indexados no Elasticsearch.

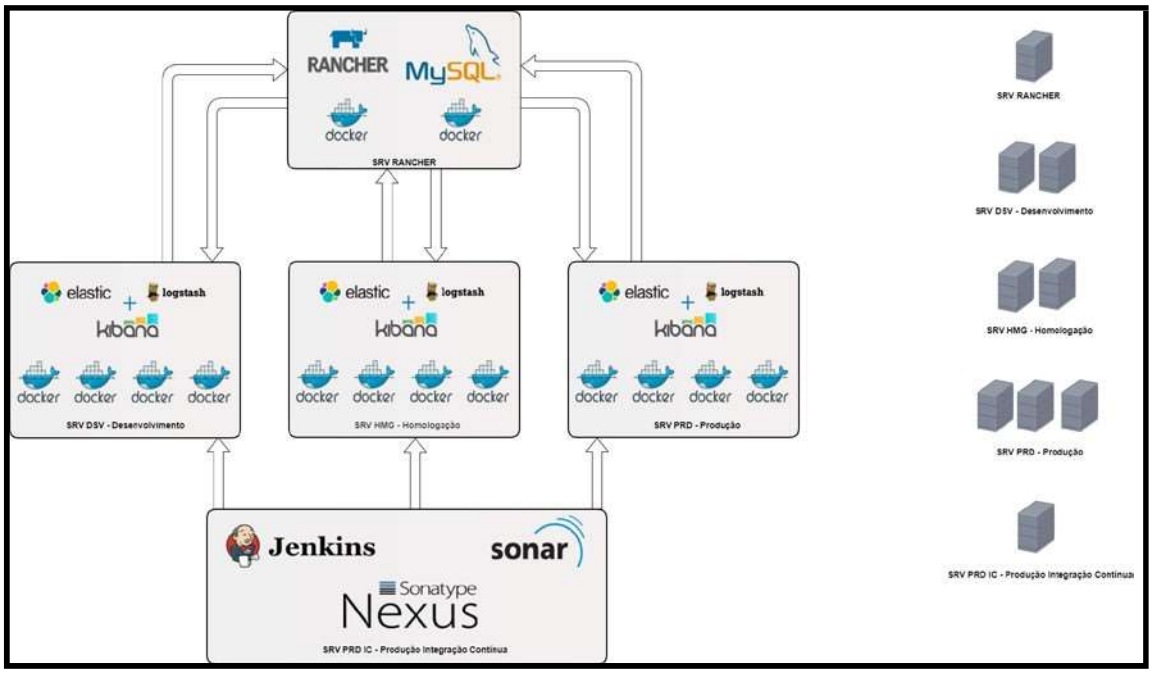


Fig.4 – Arquitetura de Infraestrutura

3. Estrutura do Docker e YML para as aplicações do MME

- Os padrões dos projetos devem seguir o seguinte modelo:
    - **<NOME_DO_SISTEMA>-WEB** e
    - **<NOME_DO_SISTEMA>-SERVICE**.
- Nas pastas “src/main/resources/config/”, caso exista a necessidade de criação de um arquivo YML, deverá existir um com o nome “application-mme.yml”, contendo as propriedades customizáveis seguindo o nome padrão “MME_NOMEVARIAVEL”;
- Cada projeto (WEB e SERVICE) deverá ter na sua estrutura a pasta Docker, contendo os arquivos necessários para o funcionamento da aplicação, exemplo:
    - sistemax-web/docker
    - sistemax-service/docker

- Dentro da pasta "docker" do SERVICE, deverão existir no mínimo 3 arquivos de configuração:
    - application.yml;
    - NOME_DO_SISTEMA_OU_MODULO.yml:
    - docker-compose.yml:

Exemplos:

application.yml 367 Bytes

```yaml
configserver:
  name: Registry
  status: Conectado na Registry MME.
jhipster:
  security:
    authentication:
      jwt:
        secret: 7UM05UM9t6f06T8O2g5awq9gsCYxTnj6LBi2jwcjkt4MFVZg5R
eureka:
    client:
        serviceUrl:
            defaultZone: http://admin:admin@VMHOST:8761/eureka/
```

sim.yml 726 Bytes

```yaml
spring:
    datasource:
        type: com.zaxxer.hikari.HikariDataSource
        url: jdbc:sqlserver://DBHOST:DBPORT;database=DBNAME;useUnicode=true;characterEncoding=UTF-8
        username: DBUSER
        password: DBPASS
    jpa:
        database-platform: org.hibernate.dialect.SQLServer2008Dialect

application:
    maximo-resultado-relatorio: 1000
    importacaoShape:
        path-files: "/uploads/"
    monitoramentoShapefiles:
        local-directory: /tmp/MME/local
    geoserver:
        workspace: GEOWS
        url: GEOHOST
        user: GEOUSER
        password: GEOPASS
    recaptcha:
        site-key: 6LeIxAcTAAAAAJcZVRqyHh71UMIEGNQ_MXjiZKhI
        secret-key: 6LeIxAcTAAAAAGG-vFI1TnRWxMZNFuojJ4WifJWe

```

docker-compose.yml 3.99 KB

```yaml
version: '2'
services:
    jhipster-registry:
        container_name: jhipster-registry
        restart: always
        environment:
            SPRING_PROFILES_ACTIVE: prod,native
            SECURITY_USER_PASSWORD: admin
            JHIPSTER_REGISTRY_PASSWORD: admin
            SPRING_CLOUD_CONFIG_SERVER_NATIVE_SEARCH_LOCATIONS: file:./central-config/
        image: jhipster/jhipster-registry:v3.2.4
        extra_hosts:
            DBHOST:  DBIP #SERVIDOR DE BANCO DE DADOS EXTERNO
        networks:
            front:
                ipv4_address: 172.36.238.10
        ports:
            - 8761:8761
        volumes:
            - /opt/registry/central-server-config:/central-config
    sim-service:
        container_name: sim-service
        hostname: sim-service
        restart: always
        image: srv099:8083/sim-service:SIMSERVICETAG
        networks:
            front:
                ipv4_address: 172.36.238.12
        links:
            - jhipster-registry:jhipster-registry
        extra_hosts:
            DBHOST:  DBIP #SERVIDOR DE BANCO DE DADOS EXTERNO
        depends_on:
            - jhipster-registry
        environment:
            SPRING_CLOUD_CONFIG_URI: http://admin:admin@VMHOST:8761/config
            SPRING_PROFILES_ACTIVE: liquibase,prod
            JHIPSTER_SLEEP: 5
        volumes:
            - /opt/sim/shapefiles:/shapefiles
    batch:
        container_name: batch
        hostname: batch
        restart: always
        image: srv099:8083/batch:BATCHTAG
        networks:
```

- Poderão existir mais de um arquivo no formato NOME_DO_SISTEMA_OU_MODULO.yml, dependendo do número de módulos existentes dentro do sistema.

- Caso, nestes arquivos acima, existam propriedades customizáveis (Ex: host, porta, banco de dados, etc . . .), estas propriedades precisam ser customizáveis, para que as mesmas possam ser alteradas em tempo de execução no momento de subirmos a imagem da aplicação no Jenkins, seguindo o nome padrão “MME_NOMEVARIAVEL”;

- Abaixo um exemplo do job do Jenkins usado para rodar as imagens das aplicações dentro do MME (em destaque as propriedades customizáveis que devem existir dentro dos arquivos yml, seguindo o nome padrão “MME_NOMEVARIAVEL”):

```
node('Build') {

    // VARIAVEIS DE AMBIENTE
    def repositorio         =  'sim-service'
    def servidor            =  'SRVZZZ'
    def servidorip          =  '127.0.0.1'

    def sim_database_ip     =  '127.0.0.1'
    def sim_database_host   =  'SRVYYY'
    def sim_database_port   =  '5678'
    def sim_database_name   =  'sim'
    def sim_database_user   =  'xxx'
    def sim_database_pass   =  'yyy'

    def ca_database_host    =  'SRVXXX'
    def ca_database_port    =  '1234'
    def ca_database_name    =  'ControleAcesso'
    def ca_database_user    =  'xxx'
    def ca_database_pass    =  'yyy'



    // CLONE DO REPOSITORIO ONDE ESTA LOCALIZADO O docker-compose.yml CRIACAO DE PASTAS
E COPIA DE ARQUIVOS
    stage('Git Clone') {
         withCredentials([
            usernamePassword(credentialsId: 'xxxx', usernameVariable: 'GITUSER',
passwordVariable: 'GITPASS'),
        ]){
           ....
        }
    }
```

```
    // ALTERACACAO DAS VARIAVEIS NO docker-compose.yml
    stage('Editando Variaveis') {
        ....
        //ALTERANDO docker-compose.yml
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBIP/${sim_database_ip}/g
/srv/dcker/docker-compose.yml'"
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBHOST/${sim_database_host}/g
/srv/dcker/docker-compose.yml'"
        sh "ssh ... usr@${servidor} 'sed -i s/MME_VMHOST/${servidorip}/g
/srv/dcker/docker-compose.yml'"
       ....

        // ALTERANDO ARQUIVO application.yml
        sh "ssh ... usr@${servidor} 'sed -i s/MME_VMHOST/${servidorip}/g
/opt/registry/application.yml'"

        // ALTERANDO ARQUIVO controleacesso.yml
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBHOST/${ca_database_host}/g
/opt/registry/controleacesso.yml'"
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBNAME/${ca_database_name}/g
/opt/registry/controleacesso.yml'"
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBUSER/${ca_database_user}/g
/opt/registry/controleacesso.yml'"
       ....

        // ALTERANDO ARQUIVO sim.yml
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBHOST/${sim_database_host}/g
/opt/registry/sim.yml'"
        sh "ssh ... usr@${servidor} 'sed -i s/MME_DBNAME/${sim_database_name}/g
/opt/registry/sim.yml'"
        ....
    }

    //REMOVE AMVBIENTE
    stage('Removendo Ambiente') {
       ....
    }

    // INICIALIZACAO DO AMBIENTE
    stage('Inicializando Ambiente') {
        input 'Deseja Inicializar o Ambiente?'
         withCredentials([
            usernamePassword(credentialsId: 'xxxx', usernameVariable: 'HUBUSER',
passwordVariable: 'HUBPASS'),
        ]){
```

```
            // LOGIN NO DOCKER REGISTRY

            sh "ssh ... usr@${servidor} 'docker login -u $HUBUSER -p $HUBPASS srv:8083'"

        }

        sh "ssh ... usr@${servidor} ' /usr/bin/docker-compose -f /srv/dcker/docker-
compose.yml up -d ; /usr/bin/docker-compose -f /srv/dcker/docker-compose.yml ps >
/tmp/ContainersRun ; cat /tmp/ContainersRun ; rm -rf /srv/docker-compose ; rm -rf
/tmp/ContainersRun'"

        echo 'SIM Iniciado, acesse em: http://sistemax.mme.gov.br'

        currentBuild.result = 'SUCCESS'

    }
}


def removeAmbiente(){

    def servidor      =  'SRVAAA'

    echo 'Removendo Ambiente em SRVAAA... '

    sh "ssh ... usr@${servidor} docker images | grep sim | awk '{print \$1\":\"\$2}' >
/tmp/images"

    sh "scp -o StrictHostKeyChecking=no -i ~/.certs-user/id_rsa /tmp/images
usr@${servidor}:/tmp/images"

    sh "ssh ... usr@${servidor} '/usr/bin/docker-compose -f /srv/dcker/docker-
compose.yml stop'"

    sh "ssh ... usr@${servidor} '/usr/bin/docker-compose -f /srv/dcker/docker-
compose.yml rm -f'"

    sh "ssh ... usr@${servidor} 'if [[ -s /tmp/images ]]; then echo Contem Imagens &&
/usr/bin/docker rmi -f \$(cat /tmp/images) && rm -rf /tmp/images && rm -rf
/tmp/checkimages && rm -rf /tmp/checkvazio; else echo Imagens nao existem && rm -rf
/tmp/images && rm -rf /tmp/checkimages && rm -rf /tmp/checkvazio; fi'"

}
```

4. Informações para as contratadas no uso da Arquitetura do MME

- As bibliotecas utilizadas para o desenvolvimento deverão possuir licenças gratuitas e fazerem parte da comunidade;
- Caso sejam utilizadas bibliotecas proprietárias da empresa que irá desenvolver a aplicação ou módulo, o MME deverá ser avisado antes de sua utilização e o código fonte da mesma deverá ser disponibilizado no GitLab do ministério, afim de ser compilada e validada dentro do órgão, esta biblioteca não poderá causar nenhum custo adicional ao ministério;
- Deverá ser usado o projeto de referência para o início do desenvolvimento, este projeto de referência está disponibilizado no GitLab do MME, antes do desenvolvimento será criado um novo repositório para o projeto a ser desenvolvido. Este sistema padrão (referência) deve ser levado em conta durante o processo de desenvolvimento, ele busca ajudar o desenvolvimento inicial e não será motivo de barreira para novas sugestões

ou melhorias, mas todas estas deverão ser analisadas antes do seu uso pela equipe do MME. A seguir os links dos projetos (Web: http://gitlab.mme.gov.br/source/sistema_padrao-web) e (Service: http://gitlab.mme.gov.br/source/sistema_padrao-service).

- Todo software e produto entregue precisa conter os seus testes e precisam ser aprovados pelas métricas definidas no Sonar do MME. Os testes devem ser executados com sucesso dentro do ambiente do MME e não apenas no ambiente da contratada.

- No jenkins, os Jobs padrões responsáveis pela compilação e deploy da aplicação (como o exemplo citado acima) serão criados em sua forma default pela equipe do MME, mas os mesmos deverão ser customizados pela fábrica de software, este trabalho é necessário para evitarmos a perda de tempo com a falta de informações para subirmos o sistema dentro da instituição. Com isso o ambiente de desenvolvimento deverá seguir o mesmo fluxo usado em todas as aplicações do MME, este fluxo já existe em todos os outros ambientes: QA, homologação e produção.