

Reunião de Orientação 03

Igor Pires dos Santos

igor.pires@ice.ufjf.br

Orientador: Rafael Bonfim



Programa de Pós-Graduação em Modelagem Computacional
Universidade Federal de Juiz de Fora

2 de outubro de 2020

Introdução

Primeiro Problema

Docker

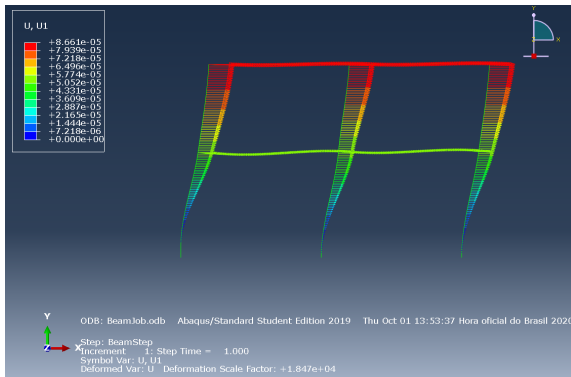
Segundo Problema

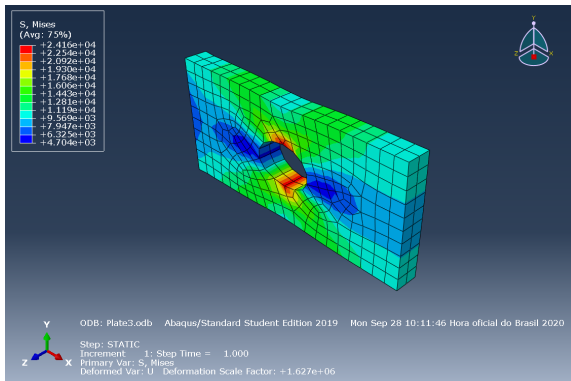
Docker Cloud Services

Terceiro Problema

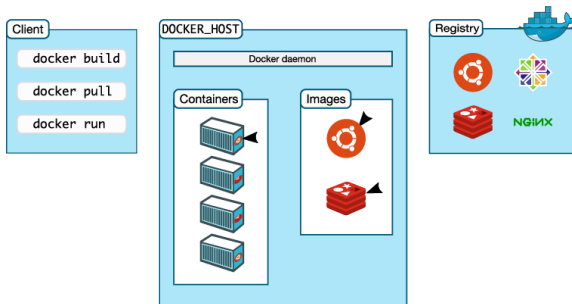
Docker Mutilayer

- ▶ **Matérias do Mestrado:** Como combinado nesse período eu foquei nas matérias restantes do mestrado, Eng. de Software e Plataformas Computacionais. Estas matérias estão acabando hoje também no dia 02/10.
- ▶ **Engenharia de Software:** Como também mencionado eu foquei no ambiente do Docker que seria utilizado para compilar o IGU em um ambiente multiplataforma.
- ▶ **Plataformas Computacionais:** Esta matéria foi feita em cima da plataforma *Abaqus CAE* que é uma ferramenta complexa utilizada na simulação de problemas mecânicos.



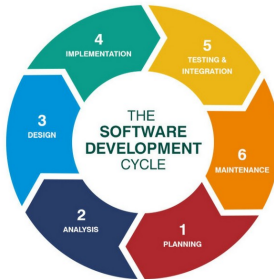


- ▶ **Engenharia de Software**
- ▶ Essa matéria foi lecionada no modelo ERE do PGMC que apresentou alguns novos desafios, por isso a matéria teve uma profundidade menor, mas o que não impediu que eu pesquisasse mais sobre o Docker.
- ▶ Como mencionado anteriormente, o Docker é uma ferramenta que se baseia em imagens, imagens estas que possuem todos os arquivos e configurações de ambiente similar à uma máquina virtual. Entretanto o Docker é muito mais poderoso pois faz isso sem o processo de virtualização, sendo muito mais rápido.
- ▶ O docker é algo que venho pesquisando desde o final do ano passado e agora finalizei um curso sobre o uso dele em conjunto com a ferramenta de CI/CD Jenkins e Atlassian Bamboo.



- ▶ Utilizando o ambiente do Docker é possível contruir essas imagens personalizadas, portanto eu construí um repositório da imagem do Ubuntu com a biblioteca Qt e as bibliotecas adicionais do OpenGL entitulado mrblackpower/ubuntuqt
- ▶ <https://hub.docker.com/repository/docker/mrblackpower/ubuntuqt>
- ▶ Equivalente ao repositório Bitbucket:
- ▶ <https://bitbucket.org/MrBlackPower/ubuntuqt/src/main/>

► O ciclo de vida de um software



Synotive

► Continuous Integration

- O processo de CI (Continuous Integration) consiste na integração de funcionalidades "recém-criadas" continuamente.
- Por exemplo, foi desenvolvida uma ferramenta que resolve o fluxo pulsátil utilizando diferentes resolvedores e compara os resultados. Um destes resolvedores ainda está em "desenvolvimento" e se deseja que esta funcionalidade seja integrada continuamente ao sistema para evitar problemas de integração.

- ▶ **O que é Integrar?**
- ▶ A definição exata do processo de integração varia de software para software, mas em geral é o processo de compilação. Em um projeto C++ comum, integrar continuamente seria executar os testes unitários e compilar o código, caso o processo falhe é gerado um alerta.

► Continuous Delivery

- O processo de CD (Continuous Delivery) consiste na entrega de funcionalidades "recém-criadas" continuamente.
- Por exemplo, a mesma ferramenta que resolvida utilizando diferentes resolvers. O resolver é integrado continuamente e se deseja que esta funcionalidade seja entregue continuamente ao cliente.

- ▶ **O que é Entregar?**
- ▶ A definição exata do processo de entrega varia de software para software, mas em geral é o processo de instalar o software na máquina. Em um projeto C++ comum, a entrega contínua seria a geração do artefato (ou executável), caso o processo falhe na criação ou inicialização é gerado um alerta.

- ▶ **O Problema:** Apesar de utilizar o mesmo SO, o mesmo compilador e as mesmas bibliotecas as versões diferiam, com isso algumas funções deixavam de ser suportadas e outras surgiam. Portanto, diversas vezes as mudanças eram criadas no programa e não era possível "integrá-las" ao ambiente externo.
- ▶ **A Solução:** Utilizar um contêiner como ambiente padrão de desenvolvimento **OU** utilizar mais de um contêiner com as versões aceitas definidas.
- ▶ **Antes:** Compilava no sistema local, rezava antes de compilar em outro lugar.

► Papel do Docker

- Conforme mencionado anteriormente o Docker é um repositório de imagens que são executadas em contêineres. Contêineres estes que podem ser executados nos mais diversos ambientes e que possuem definições próprias do seu ambiente interno. Portanto, através do Docker é possível se padronizar o ambiente em que esta integração e entrega ocorrerão.

- ▶ O mais interessante do Docker é que ele funciona em conjunto com as tecnologias já existentes para CI/CD, podendo ser utilizado em partes ou em todo o processo.
- ▶ Com ferramentas automatizadoras como o Jenkins e o Atlassian Bamboo é possível utilizar contêineres internas à estas ferramentas que executarão os testes, compilarão o código, armazenarão os artefatos e entregarão estes artefatos.



Jenkins

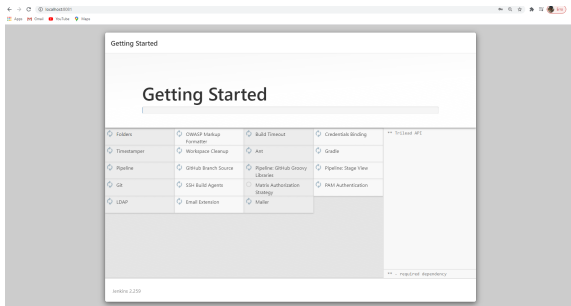
- ▶ Aplicar o conceito de CI & CD efetivamente é realizar todos os passos que você (o programador, o analista de sistema ou devOps) executa após terminar a edição do código de forma automática e/ou periódica.
- ▶ Assim como o git, o Bamboo e o Jenkins são servidores que rodam continuamente. Estes automatizadores recebem sinais de "gatilhos" (*hooks*) ou realizam chamadas periodicamente. Estes servidores são capazes de utilizar as tecnologias de repositório do git e a contêinerização do docker.
- ▶ Isto é, ao enviar o código para o repositório um gatilho é ativo e recebido pelo automatizador, que por sua vez envia o código fonte aos contêineres e realiza as ações através de comandos CLI ou Powershell. É possível ainda utilizar protocolos adjacentes como SSH e o SCP que permitem a utilização de recursos na mesma rede.

- ▶ Ambas a ferramentas de CI/CD aqui citadas foram desenvolvidas em Java utilizando Apache Tomcat e JDK, sendo o Jenkins Open Source. Logo, ambas as soluções podem ser utilizadas sem o Docker.
- ▶ Entretanto, é extremamente fácil utilizar estas ferramentas em união com o Docker. Ambas a ferramentas possuem imagens no repositório com os servidores e podem ser iniciadas com:
- ▶ *docker container run atlassian/bamboo-server*
- ▶ *docker container run jenkins/jenkins*

- ▶ Com estes comandos os servidores são criados, entretanto é necessária a configuração dos *volumes* para que os dados configurados sejam persistidos em disco corretamente.

```
jenkins:
  container_name: jenkins
  image: jenkins/jenkins
  ports:
    - "8081:8080"
  volumes:
    - "./jenkins:/var/jenkins_home"
  networks:
    - net

bamboo:
  container_name: bamboo
  image: mrblackpower/bamboo-server
  build:
    context: bamboo_build
  ports:
    - "54663:54663"
    - "8085:8085"
  volumes:
    - "./bamboo:/var/atlassian/application-data/bamboo"
  networks:
    - net
```

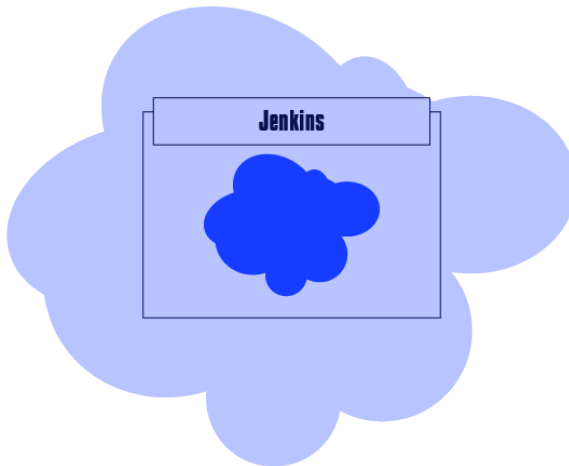


- ▶ Com isto os servidores rodam e têm suas portas expostas. Assim como um servidor git os servidores de CI/CD possuem projetos e contas de usuários. Estas contas são utilizadas pelos servidores para gerenciar qual usuário tem acesso à qual projeto e quais permissões sobre o projeto.
- ▶ Isto é extremamente útil para empresas e softwares maiores, pois um processo de deployment significaria uma mudança em servidor que não poderia e nem deveria ser feita por um grande número de pessoas.

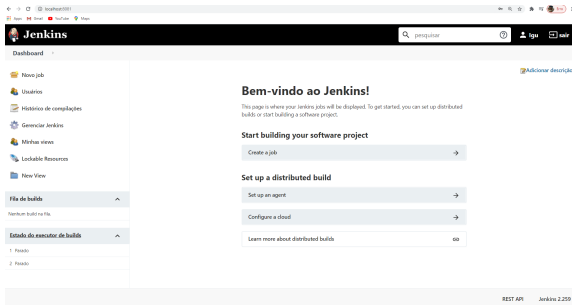
- ▶ **O Problema:** Após o processo de compilação gostaria de se entregar efetivamente o executável gerado.
- ▶ **A Solução:** Compilar o código-fonte dentro do contêiner e enviar artefato à nuvem **OU** envia artefatos para outro lugar.
- ▶ **Antes:** Mesmo processo, entretanto manualmente.

- ▶ Utilizando imagens Unix é possível instalar os serviços AWS (*Amazon Web Services*) e GCP (*Google Cloud Platform*) que possibilitam o envio destes artefatos à nuvem instantaneamente.
- ▶ A grande vantagem de utilizar estes serviços da nuvem é que eles podem ser consumidos de qualquer lugar com acesso à internet.

- ▶ **O Problema:** Tá, fiz isso tudo, mas agora quero fazer multiplataforma (Linux + Window).
- ▶ **A Solução:** Utilizar Docker in Docker **OU** Docker beside Docker com as imagens de SO desejadas.
- ▶ **Antes:** Se você compilar roda...



- ▶ Docker beside Docker consistem em utilizar contêineres docker ao lado do contêiner que contém o servidor de CI/CD. Desta forma os comandos são executados internamente dos contêineres adjacentes através da rede definida entre eles utilizando os protocolos já conhecidos, como SSH e SCP.
- ▶ É possível utilizar esta tecnologia também sem utilizar o servidor CI/CD dentro de um contêiner mas localmente. Basta abrir as portas locais corretamente e configurar o servidor CI/CD com o endereço local ao invés do endereço do contêiner.




► Plugins


Git	✓ Sucesso
SSH Build Agents	✓ Sucesso
Matrix Authorization Strategy	✓ Sucesso
PAM Authentication	✓ Sucesso
LDAP	✓ Sucesso
Email Extension	✓ Sucesso
Mailer	✓ Sucesso
Loading plugin extensions	✓ Sucesso
SSH	✓ Sucesso
SSH Agent	✓ Sucesso
SSH Pipeline Steps	✓ Sucesso
Loading plugin extensions	✓ Sucesso
Multiple SCMs	✓ Sucesso
Mercurial	✓ Sucesso
Bitbucket Build Status Notifier	✓ Sucesso
Generic Webhook Trigger	✓ Sucesso
Bitbucket OAuth	✓ Sucesso
Job DSL	✓ Sucesso
Bitbucket Push and Pull Request	✓ Sucesso
Bitbucket Server Integration	✓ Sucesso
Bitbucket	✓ Sucesso
Bitbucket Server Notifier	✓ Sucesso
Loading plugin extensions	⚙ Running
Reiniciando o Jenkins	⚙ Pendente

Enter an item name


» Required field

**Construir um projeto de software free-style**


Esta é a central de funcionalidades do Jenkins. Ele construirá seu projeto, e você pode combinar qualquer SCM com qualquer sistema de builds, e ele até mesmo pode ser usado para outras jobs diferentes de builds de software.

**Pipeline**


Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Construir projeto de múltiplas configurações**

Apropriado para projetos que necessitam de grande número de diferentes configurações, como teste em múltiplos ambientes, builds para plataformas específicas, etc.

**Folder**

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

**GitHub Organization**

Scans a GitHub organization (or user account) for all repositories matching some defined markers.

General **Gerenciamento de código fonte** Trigger de builds Ambiente de build Build Ações de pós-build

Gerenciamento de código fonte

☐ Nenhum

☒ Bitbucket Server

Credentials (for build auth)

Bitbucket Server instance

Project name
Using 'Alexandria' at http://bitbucket:7990/projects/ALE

Repository name
Using 'IGU' at http://bitbucket:7990/projects/ALE/repos/igu/browse

Clone from

Optional SSH Credentials

Branches to build

Branch Specifier (blank for 'any')

General **Gerenciamento de código fonte** Trigger de builds Ambiente de build Build Ações de pós-build

Gerenciamento de código fonte

☐ Nenhum
☒ Bitbucket Server

Credentials (for build auth)

Bitbucket Server instance

Project name
Using 'Alexandria' at http://bitbucket:7990/projects/ALE

Repository name
Using 'IGU' at http://bitbucket:7990/projects/ALE/repos/igu/browse

Clone from

Optional SSH Credentials

Branches to build

Branch Specifier (blank for 'any')

- ▶ Além de toda essa configuração é necessário também configurar o servidor SSH rodando no contêiner de destino, tal configuração é feita na construção do contêiner.
- ▶ O problema com Docker beside Docker é que contêineres Windows não rodam com o mesmo suporte que contêineres Unix, isto é, é possível rodar contêineres Unix ou contêineres Windows, mas não ambos ao mesmo tempo.

- ▶ Docker in Docker consistem em utilizar contêineres docker dentro do contêiner que contém o servidor de CI/CD. Desta forma os comandos são executados internamente dos contêineres.
- ▶ É possível utilizar esta tecnologia também sem utilizar o servidor CI/CD dentro de um contêiner mas localmente. Basta ter o Docker e o servidor de CI/CD instalado no mesmo ambiente.

- ▶ (mostrar ambiente com Jenkins e Bamboo)
- ▶ Novos paradigmas, novos problemas!

► Problemas:

- Compilar em um mesmo computador contêineres Windows e Unix está se provando cada vez mais difícil, entretanto descobri que não é necessariamente preciso, porque do linux tem como compilar para o Windows, mas é igualmente complicado (Applimages).
- Mas é possível disponibilizar uma imagem com o IGU instalado e sendo utilizado através da linha de comando.

► **Proposta Anterior**

- Utilizar o IGU "1.0" para realizar os estudos de caso
- Criar as ferramentas do IGU "2.0" (as mesmas presentes no IGU "1.0" adicionadas ao paradigma Paralelismo, Interface Atualizada, Linha de Comando, Programa Distribuído).

► Proposta Atual

- Outubro: Realização do Toefl iBT, Adicionar possibilidade de se executar o IGU 1.0 via linha de comando, Finalizar Repositório de imagem com o IGU 1.0 Instalado, Definir os experimentos relevantes com CCO (Ye, Pressão e Fluxo?)(Quais parâmetros ?), Definir se um sistema de CI/CD será aplicado e QUAL e COMO.
- Novembro: Finalização da Graduação, Rodar os experimentos relevantes com CCO utilizando o IGU 1.0 , escrever (e plotar) estes resultados.
- Dezembro: Finalizar parte Gráfica IGU 2.0, finalizar sistema de CI/CD, re-analisar funcionalidades do IGU 2.0 à adicionar e começar dissertação.
- Janeiro: Começar IGU 2.0 e sua parte distribuída e escrever dissertação.
- Fevereiro: Finalizar IGU 2.0 e sua parte distribuída, extrair resultados e escrever dissertação.

- ▶ Março: Escrever dissertação.
- ▶ Abril: ??
- ▶ Maio: ??
- ▶ Junho: ??

- ▶ Dúvidas:
- ▶ Qual a linha de pesquisa eu devo procurar quando estiver procurando instituições da Alemanha de pesquisa?