Artigo
Danusa Calixto · Nov. 9, 2022 9min de leitura

Entrega contínua de sua solução InterSystems usando GitLab – Parte II: Fluxo de trabalho do GitLab

Nesta série de artigos, quero apresentar e discutir várias abordagens possíveis para o desenvolvimento de software com tecnologias da InterSystems e do GitLab. Vou cobrir tópicos como:

- Git básico
- Fluxo Git (processo de desenvolvimento)
- Instalação do GitLab
- **Fluxo de trabalho do GitLab**
- **Entrega contínua**
- Instalação e configuração do GitLab
- CI/CD do GitLab

No artigo anterior, abordamos os fundamentos do Git, por que um entendimento de alto nível dos conceitos do Git é importante para o desenvolvimento de software moderno e como o Git pode ser usado para desenvolver software. Ainda assim, nosso foco foi na parte da implementação do desenvolvimento de software, mas esta parte apresenta:

- **Fluxo de trabalho do GitLab** — um processo completo do ciclo de vida do software, desde a ideia até o feedback do usuário
- **Entrega Contínua** — uma abordagem de engenharia de software em que as equipes produzem software em ciclos curtos, garantindo que o software possa ser lançado de forma confiável a qualquer momento. Seu objetivo é construir, testar e lançar software com mais rapidez e frequência.

Fluxo de trabalho do GitLab

O fluxo de trabalho do GitLab é uma sequência lógica de possíveis ações a serem tomadas durante todo o ciclo de vida do processo de desenvolvimento de software.

O fluxo de trabalho do GitLab leva em consideração o fluxo do GitLab, que discutimos em um artigo anterior. Veja como funciona:

1. Ideia: todas as novas propostas começam com uma ideia.
2. Problema: a maneira mais eficaz de discutir uma ideia é criar um problema para ela. Sua equipe e seus colaboradores podem ajudar você a aprimorar e melhorar a ideia no rastreador de problemas.
3. Plano: quando a discussão chega a um acordo, é hora de programar. Porém, primeiro, precisamos priorizar e organizar nosso fluxo de trabalho ao atribuir problemas a marcos e quadro de problemas.
4. Código: agora estamos prontos para escrever nosso código, já que está tudo organizado.
5. Commit: depois de satisfeitos com o rascunho, podemos enviar nosso código para um feature-branch com controle de versão. O fluxo do GitLab foi explicado em detalhes no artigo anterior.
6. Teste: executamos nossos scripts usando o CI GitLab, para construir e testar nosso aplicativo.
7. Revisão: assim que nosso script funcionar e nossos testes e compilações forem bem-sucedidos, estamos prontos para que nosso código seja revisado e aprovado.
8. Staging: agora é hora de implantar nosso código em um ambiente de staging para verificar se tudo funciona como esperado ou se ainda precisamos de ajustes.
9. Produção: quando tudo estiver funcionando como deve, é hora de implantar no nosso ambiente de

produção!

10. Feedback: agora é hora de olhar para trás e verificar qual etapa do nosso trabalho precisa ser melhorada.

Novamente, o processo em si não é novo (ou exclusivo do GitLab) e pode ser alcançado com outras ferramentas da sua escolha.

Vamos discutir várias dessas etapas e o que elas implicam. Também há documentação disponível.

Problema e plano

As etapas iniciais do fluxo de trabalho do GitLab são centradas em um **problema**: um recurso, bug ou outro tipo de trabalho semanticamente separado.

O problema tem várias finalidades, como:

- Gerenciamento: um problema tem data de vencimento, pessoa designada, tempo gasto e estimativas, etc. para ajudar a monitorar a resolução do problema.
- Administrativo: um problema faz parte de um marco, quadro kanban, que nos permite rastrear nosso software à medida que ele avança de versão para versão.
- Desenvolvimento: um problema tem uma discussão e commits associados a ele.

A etapa de planejamento nos permite agrupar os problemas por prioridade, marco, quadro kanban e ter uma visão geral disso.

O desenvolvimento foi discutido na parte anterior, basta seguir qualquer fluxo git que quiser. Depois que desenvolvemos nosso novo recurso e o mesclamos no master: o que vem depois?

Entrega contínua

A entrega contínua é uma abordagem de engenharia de software em que as equipes produzem software em ciclos curtos, garantindo que o software possa ser lançado de forma confiável a qualquer momento. Seu objetivo é construir, testar e lançar software com mais rapidez e frequência. A abordagem ajuda a reduzir o custo, o tempo e o risco da entrega de alterações, permitindo mais atualizações incrementais para aplicativos em produção. Um processo de implantação simples e repetível é importante para a entrega contínua.

Entrega contínua no GitLab

No GitLab, a configuração da entrega contínua é definida por repositório como um arquivo de configuração YAML.

- A configuração de entrega contínua é uma série de **estágios** consecutivos.
- Cada estágio tem um ou vários **scripts** que são executados em paralelo.

O script define uma ação e quais condições devem ser atendidas para executá-la:

- O que fazer (executar o comando do SO, executar um contêiner)?
- Quando executar o script:
  - Quais são os gatilhos (commit de um branch específico)?
  - Nós o executamos se os estágios anteriores falharam?
- Executar manualmente ou automaticamente?

- Em que ambiente executar o script?
- Quais artefatos salvar após a execução dos scripts (eles são carregados do ambiente para o GitLab para facilitar o acesso)?

Ambiente - é um servidor ou contêiner configurado no qual você pode executar seus scripts.

Runners executam scripts em ambientes específicos. Eles são conectados ao GitLab e executam scripts conforme necessário.

O runner pode ser implantado em um servidor, contêiner ou até mesmo na sua máquina local.

Como acontece a entrega contínua?

1. O novo commit é enviado para o repositório.
2. O GitLab verifica a configuração de entrega contínua.
3. A configuração de entrega contínua contém todos os scripts possíveis para todos os casos, para que sejam filtrados para um conjunto de scripts que devem ser executados para esse commit específico (por exemplo, um commit para o branch master aciona apenas ações relacionadas a um branch master). Esse conjunto é chamado de **pipeline**.
4. O pipeline é executado em um ambiente de destino e os resultados da execução são salvos e exibidos no GitLab.

Por exemplo, aqui está um pipeline executado após um commit em um branch master:

Ele consiste em quatro etapas, executadas consecutivamente

1. O estágio de carregamento carrega o código em um servidor
2. O estágio de teste executa testes de unidade
3. O estágio de pacote consiste em dois scripts executados em paralelo:
    - Compilação cliente
    - Código de exportação do servidor (principalmente para fins informativos)
4. O estágio de implantação move o cliente criado para o diretório do servidor web.

Como podemos ver, todos os scripts foram executados com sucesso. Se um dos scripts falhar, por padrão, os scripts posteriores não são executados (mas podemos alterar esse comportamento):

Se abrirmos o script, podemos ver o log e determinar por que ele falhou:

```
Running with gitlab-runner 10.4.0 (857480b6)
 on test runner (ab34a8c5)
Using Shell executor...
Running on gitlab-test...
&lt;span class="term-fg-l-green term-bold">Fetching changes...&lt;/span>
Removing diff.xml
Removing full.xml
Removing index.html
Removing tests.html
HEAD is now at a5bf3e8 Merge branch '4-versiya-1-0' into 'master'
From http://gitlab.eduard.win/test/testProject
 * [new branch] 5-versiya-1-1 -> origin/5-versiya-1-1
 a5bf3e8..442a4db master -> origin/master
 d28295a..42a10aa preprod -> origin/preprod
 3ac4b21..7edf7f4 prod -> origin/prod
&lt;span class="term-fg-l-green term-
bold">Checking out 442a4db1 as master...&lt;/span>
&lt;span class="term-fg-l-green term-bold">Skipping Git submodules setup&lt;/span>
```

```
&lt;span class="term-fg-l-green term-
bold">$ csession ensemble "##class(isc.git.GitLab).loadDiff()"&lt;/span>

[2018-03-06 13:58:19.188] Importing dir /home/gitlab-
runner/builds/ab34a8c5/0/test/testProject/

[2018-03-06 13:58:19.188] Loading diff between a5bf3e8596d842c5cc3da7819409ed81e62c31
e3 and 442a4db170aa58f2129e5889a4bb79261aa0cad0

[2018-03-06 13:58:19.192] Variable modified
var=$lb("MyApp/Info.cls")

Load started on 03/06/2018 13:58:19
Loading file /home/gitlab-
runner/builds/ab34a8c5/0/test/testProject/MyApp/Info.cls as udl
Load finished successfully.

[2018-03-06 13:58:19.241] Variable items
var="MyApp.Info.cls"
var("MyApp.Info.cls")=""

Compilation started on 03/06/2018 13:58:19 with qualifiers 'cuk /checkuptodate=expand
edonly'
Compiling class MyApp.Info
Compiling routine MyApp.Info.1
ERROR: MyApp.Info.cls(version+2) #1003: Expected space : '}' : Offset:14 [zversion+1^
MyApp.Info.1]
 TEXT:  quit, "1.0" }
Detected 1 errors during compilation in 0.010s.

[2018-03-06 13:58:19.252] ERROR #5475: Error compiling routine: MyApp.Info.1. Errors:
 ERROR: MyApp.Info.cls(version+2) #1003: Expected space : '}' : Offset:14 [zversion+1
^MyApp.Info.1]
 > ERROR #5030: An error occurred while compiling class 'MyApp.Info'
&lt;span class="term-fg-l-red term-bold">ERROR: Job failed: exit status 1
&lt;/span>
```

O erro de compilação causou a falha do nosso script.

Conclusão

- O GitLab é compatível com todos os principais estágios de desenvolvimento de software.
- A entrega contínua pode ajudar você a automatizar tarefas de construção, teste e implantação do seu software.

Links

- Parte I: Git
- Introdução ao fluxo de trabalho do GitLab
- Documentação de CI/CD do GitLab
- Fluxo do GitLab
- Código deste artigo

O que vem a seguir?

No próximo artigo, vamos:

- Instalar o GitLab.
- Conectá-lo a diversos ambientes com os produtos InterSystems instalados.
- Escrever uma configuração de entrega contínua.

Vamos discutir como a entrega contínua deve funcionar.

Em primeiro lugar, precisamos de vários ambientes e branches que correspondam a eles. O código entra nesse branch e é entregue ao ambiente de destino:

Ambiente	Branch	Entrega	Quem pode fazer envios	Quem pode mesclar
Teste	master	Automático	Desenvolvedores Proprietários	Desenvolvedores Proprietários
Preprod	preprod	Automático	Ninguém	Proprietários
Prod	prod	Semiautomático (pressionar botão para entregar)	Ninguém	Proprietários

E, como exemplo, desenvolveremos um novo recurso usando o fluxo do GitLab e o entregaremos usando a CD do GitLab.

1. O recurso é desenvolvido em um branch de recursos.
2. O branch de recurso é revisado e mesclado no master branch.
3. Depois de um tempo (vários recursos mesclados), o master é mesclado com o preprod
4. Depois de um tempo (teste do usuário, etc.), o preprod é mesclado com o prod

Veja como ficaria:

1. Desenvolvimento e teste
    - O desenvolvedor envia o código para o novo recurso em um branch de recursos separado
    - Depois que o recurso se torna estável, o desenvolvedor mescla nosso branch de recursos no master branch
    - O código do branch master é entregue ao ambiente de teste, onde é carregado e testado
2. Entrega para o ambiente de pré-produção
    - O desenvolvedor cria a solicitação de mesclagem do branch master para o branch de pré-produção
    - Depois de algum tempo, o proprietário do repositório aprova a solicitação de mesclagem
    - O código do branch de pré-produção é entregue ao ambiente de pré-produção
3. Entrega para o ambiente de produção
    - O desenvolvedor cria a solicitação de mesclagem do branch de pré-produção para o branch de produção
    - Depois de algum tempo, o proprietário do repositório aprova a solicitação de mesclagem
    - O proprietário do repositório aperta o botão "Implantar"
    - O código do branch de produção é entregue ao ambiente de produção

Ou o mesmo, mas em formato gráfico:

#Administração do Sistema #Conteinerização #Docker #Gestão da Mudança #Git #Implantação #Iniciante #Integração Contínua #Melhores Práticas #Caché