**ANEXO IV**

**Ambiente Conteiner – Devops do MMA**

# Introdução

O presente documento apresenta os detalhes da esteira CD/CD implementada no projeto DevOps do MMA. Essa nota técnica contempla todos os passos desde a criação dos repositórios, ambientes no Openshift até exemplos dos artefatos necessários. O documento está dividido na descrição dos seguintes itens

* Documentação do Processo
* Detalhamento da esteira
* Exemplos de Artefatos
* Criação dos ambientes no Jenkins X
* Permissionamento de namespaces no Openshift
* Repositório de Documentação

# Documentação do Processo

Segue um diagrama detalhando o processo para novas aplicações.

Destaca-se:

* Interação das fábricas diretamente com a CGTI.
* Registro de chamado para criação dos acessos e repositórios no GITLAB/Rede do MMA

Repositório da documentação: **gitops.mma.com.br/documentacao/APP**

Repositórios dos componentes: **gitops.mma.gov.br/sistemas/APP**

Repositórios dos ambientes: **gitops.mma.gov.br/infra/environment-jx-APP-ENV**

* Construção dos artefatos sendo realizadas pelas equipes de desenvolvimento com apoio da equipe de infraestrutura.
* Disponibilização da aplicação no ambiente de Desenvolvimento do MMA.

# Pré-requisitos da aplicação

Para o correto funcionamento na esteira e melhor gestão durante operação, deve-se considerar as seguintes características:

As aplicações na esteira irão ser executadas na plataforma Kubernetes/Openshift, que se beneficia com aplicações modernas que seguem os princípios dos 12 fatores. A metodologia doze-fatores pode ser aplicada a aplicações escritas em qualquer linguagem de programação, e que utilizem qualquer combinação de serviços de suportes (banco de dados, filas, cache de memória etc.).

Segue breve descrição de algumas características de aplicação que segue os princípios de 12 fatores:

OBS: É recomendo a leitura do material na íntegra em: **https://12factor.net/pt\_br/**

* **Repositórios separados por componentes.**

Exemplo: aplicação agatha possui componente: frontend e backend. Cada componente terá seu próprio repositório com seus próprios artefatos. (sistemas/agatha-frontend e sistemas/agatha-backend)

* **Usar variáveis de ambientes ou arquivos de configuração externos.**

As informações passadas para aplicação como senha de banco, usuário de conexão, propriedades da aplicação que são específicas por ambientes, devem ser externalizadas da aplicação, ou seja, **não devem** depender de build para alterar os valores.

### **Usar serviços de apoio como recursos anexados**

O código para um app doze-fatores não faz distinção entre serviços locais e de terceiros. Para o app, ambos são recursos anexados, acessíveis via uma URL ou outro localizador/credenciais na [config](https://12factor.net/pt_br/config). Recursos podem ser anexados e desanexados a deploys à vontade. Por exemplo, se o banco de dados do app não está funcionando corretamente devido a um problema de hardware, o administrador do app pode subir um novo servidor de banco de dados restaurado de um backup recente. O atual banco de produção pode ser desanexado, e o novo banco anexado – tudo sem nenhuma mudança no código.

### **Separe estritamente os estágios de construção e execução**

Há separação estrita entre os estágios de construção, lançamento e execução. Por exemplo, é impossível alterar código em tempo de execução, já que não há meios de se propagar tais mudanças de volta ao estágio de construção.

### **Execute a aplicação como um ou mais processos que não armazenam estado**

A aplicação doze-fatores nunca assume que qualquer coisa cacheada na memória ou no disco estará disponível em uma futura solicitação ou job – com muitos processos de cada tipo rodando, as chances são altas de que uma futura solicitação será servida por um processo diferente. Mesmo quando rodando em apenas um processo, um restart (desencadeado pelo deploy de um código, mudança de configuração, ou o ambiente de execução realocando o processo para uma localização física diferente) geralmente vai acabar com todo o estado local (por exemplo, memória e sistema de arquivos).

### **Exporte serviços via vínculo de portas**

O aplicativo doze-fatores é completamente autocontido e não depende de injeções de tempo de execução de um servidor web em um ambiente de execução para criar um serviço que defronte a web. O app web exporta o HTTP como um serviço através da vinculação a uma porta, e escuta as requisições que chegam na mesma. Note que a abordagem de vincular portas significa que um app pode se tornar o [serviço de apoio](https://12factor.net/pt_br/backing-services) para um outro app, provendo a URL do app de apoio como um identificador de recurso na [configuração](https://12factor.net/pt_br/config) para o app consumidor.

### **Escale através do processo modelo**

O desenvolvedor pode arquitetar a aplicação para lidar com diversas cargas de trabalho, atribuindo a cada tipo de trabalho a um *tipo de processo*. Por exemplo, solicitações HTTP podem ser manipuladas para um processo web, e tarefas background de longa duração podem ser manipuladas por um processo trabalhador. Podendo **escalar** sem significativas mudanças em ferramentas, arquiteturas, ou práticas de desenvolvimento.

### **Maximize robustez com inicialização rápida e desligamento gracioso**

Os [processos](https://12factor.net/pt_br/processos) de um app doze-fatores são *descartáveis*, significando que podem ser iniciados ou parados a qualquer momento. Isso facilita o escalonamento elástico, rápido deploy de [código](https://12factor.net/pt_br/codebase) ou mudanças de [configuração](https://12factor.net/pt_br/config), e robustez de deploys de produção.

### **Mantenha o desenvolvimento, homologação e produção o mais similares possíveis**

O App doze-fatores é projetado para implantação contínua, deixando a lacuna entre desenvolvimento e produção pequena.

### **Trate logs como fluxos de eventos**

Um app doze-fatores nunca se preocupa com o roteamento ou armazenagem do seu fluxo de saída. Ele não deve tentar escrever ou gerir arquivos de logs. No lugar, cada processo em execução escreve seu próprio fluxo de evento, sem buffer, para o **stdout**.

# Detalhamento da Esteira

Todo o fluxo da esteira é controlado pelo Jenkinsfile que deverá estar na raiz de cada repositório da aplicação. (O repositório gitops.mma.gov.br/sistemas/artefatos-cicd exemplifica os artefatos com mais detalhes).

Usando a filosofia GITOPS, para cada aplicação teremos:

* 1 repositório por componente. (Exemplo: sistemas/agatha)
* 1 repositório para cada ambiente.

Exemplo:

* **infra/environment-jx-agatha-dev,**

Representa o repositório de código da infraestrutura da aplicação agatha no **ambiente de desenvolvimento**

* infra/environment-jx-agatha-hmg,

Representa o repositório de código da infraestrutura da aplicação agatha no **ambiente de homologação**

* infra/environment-jx-agatha-prod.

Representa o repositório de código da infraestrutura da aplicação agatha no **ambiente de produção**

Com todos os artefatos criados e funcionais (Detalhes nos tópicos abaixo):

1 - A esteira inicia com qualquer evento na branch: **develop**.

2 - O gitlab via webhook notifica o jenkins e inicia a esteira.

3 - O jenkins interpreta o conteúdo do Jenkinsfile disponível no repositório da aplicação.

4 - O Jenkins inicia um POD para executar esse job, o POD depende do informado no Jenkinsfile, por exemplo: Para aplicações Java, usado o POD: jenkins-maven.

5 - Dentre as atividades que executa, é gerado de forma automática uma TAG semântica, com sufixo **-dev** usando o comando do jenkinsx: jx-release-version, essa tag será usada para rastreabilidade do código no:

* gitlab
* sonar
* ambientes

6 - Realiza atividades de scanner no código e build se necessário (arquivo.ear/war).

7 - Constrói a imagem docker final usando o Dockerfile da aplicação e publica no docker registry interno do cluster.

8 -Empacota a aplicação usando o helm chart que deve estar no código e publica no repositório local: chartmuseum.

9 - Executa o comando do jenkinsx: jx promote, que publica a aplicação com a versão especificada no ambiente informado. No caso do Jenkinsfile, ele publica a aplicação recém criada automaticamente no ambiente de dev.

Todos os passos acima são similares para o ambiente de hmg (branch homolog) e prod (branch master)

Nos repositórios dos ambientes é controlado qual versão da aplicação deve estar aplicada.

Exemplo:

Aplicação: **Agatha-Backend**

Repositórios:

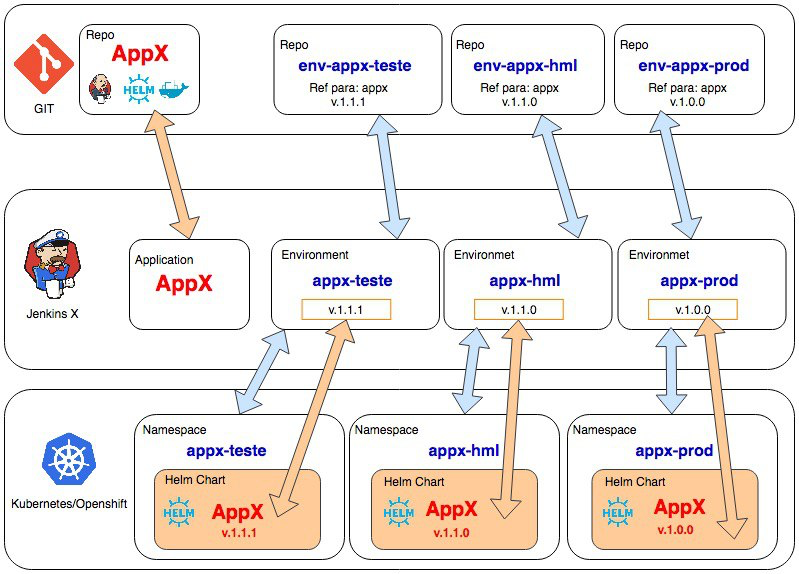
App: <http://gitops.mma.gov.br/sistemas/agatha-backend>

Dev: <http://gitops.mma.gov.br/infra/environment-jx-agatha-dev>

Hmg: <http://gitops.mma.gov.br/infra/environment-jx-agatha-hmg>

Prod: <http://gitops.mma.gov.br/infra/environment-jx-agatha-prod>

Segue diagrama representando os repositórios e as versões criadas durante a esteira e os namespaces no OKD:



Ao incluir um novo código na branch develop do repositório da agatha-backend a esteira inicia.

* Cria uma tag incremental, constrói a imagem e o helm chart usando a tag criada.

Exemplo: 0.0.18-dev

* Publica no docker registry e no chartmuseum

Exemplo: Tag da imagem docker: 0.0.18-dev, versão do helm chart: 0.0.18-dev

* Atualiza o arquivo: env/requirements.yaml no código que representa o ambiente de DEV, com o comando jx promote

env/requirements.yaml (<http://gitops.mma.gov.br/infra/environment-jx-agatha-dev>)

…..

dependencies:

- name: agatha-backend

repository: http://jenkins-x-chartmuseum:8080

version: 0.0.18-dev

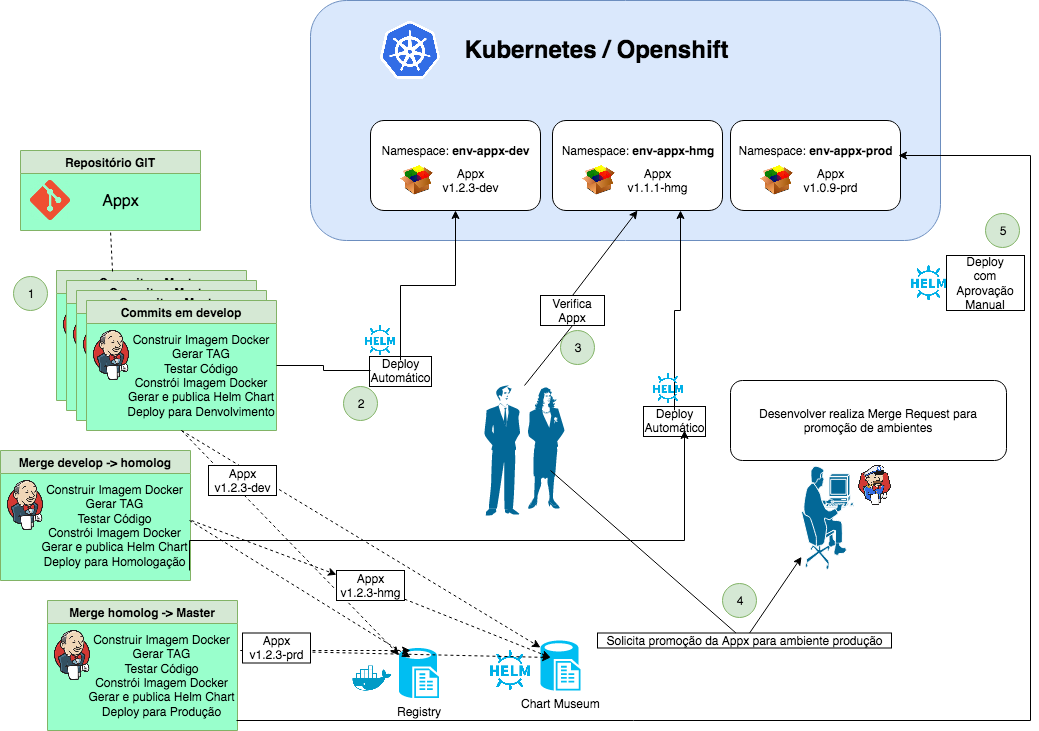
O processo de promoção é realizando um Merge Request de:

Promoção para homologação: **develop** para **homolog**

Promoção para Produção: **homolog**  para master.

OBS: A promoção em produção por questões de maior controle, não é totalmente automática, ao executar o Jenkinsfile na branch master, ainda é preciso acessar o código do ambiente de prod e aprovar manualmente um Merge Request nesse ambiente.

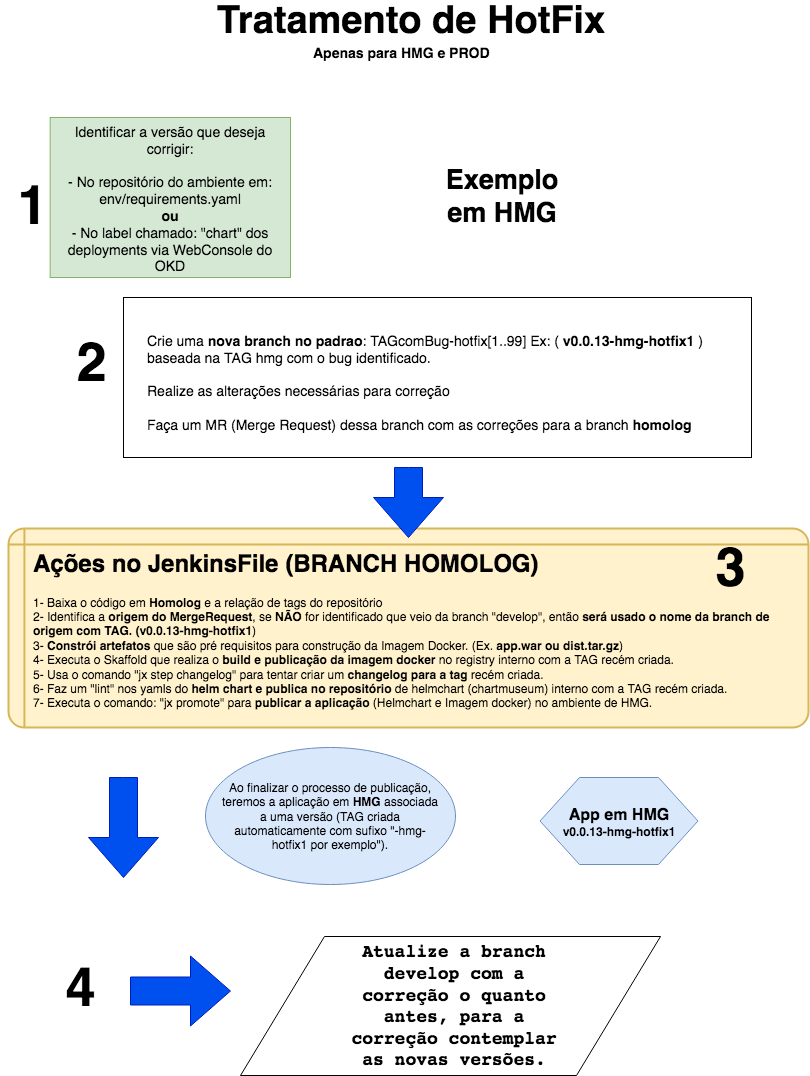
Segue diagrama exemplificando o fluxo durante operação



**Segue diagrama com ações realizadas pelo Jenkinsfile e detalhes da promoção para o fluxo padrão:**



Segue diagrama para tratamento de exceções ao fluxo padrão:



# Criando repositório para novas Apps no GITLAB

**[Atividade exclusiva da equipe de infraestrutura]**

Na inclusão de uma nova aplicação/componente na esteira é preciso criar o repositório no GITLAB. O repositório deve estar no grupo: **sistemas** do GITLAB e possuir as branchs padrões da esteira:

* develop
* homolog
* master

As atividades são:

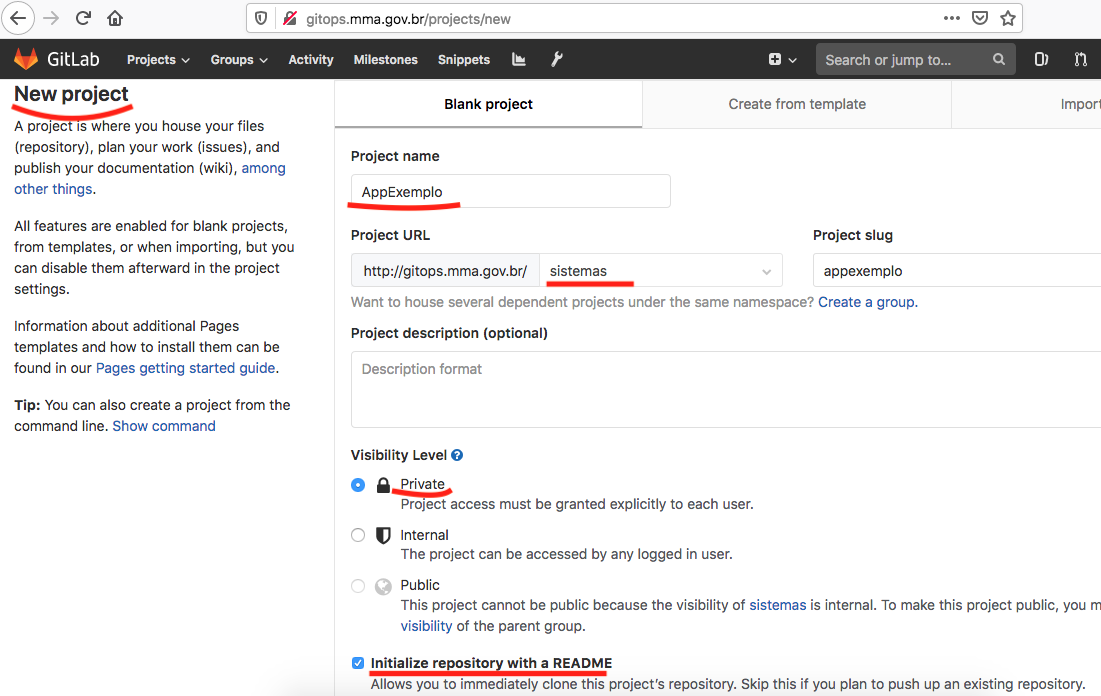
1 - Criar o repositório abaixo do grupo **sistemas**

2 - Criar as branchs que a esteira usa

3 - Definir a branch **develop**  como branch default

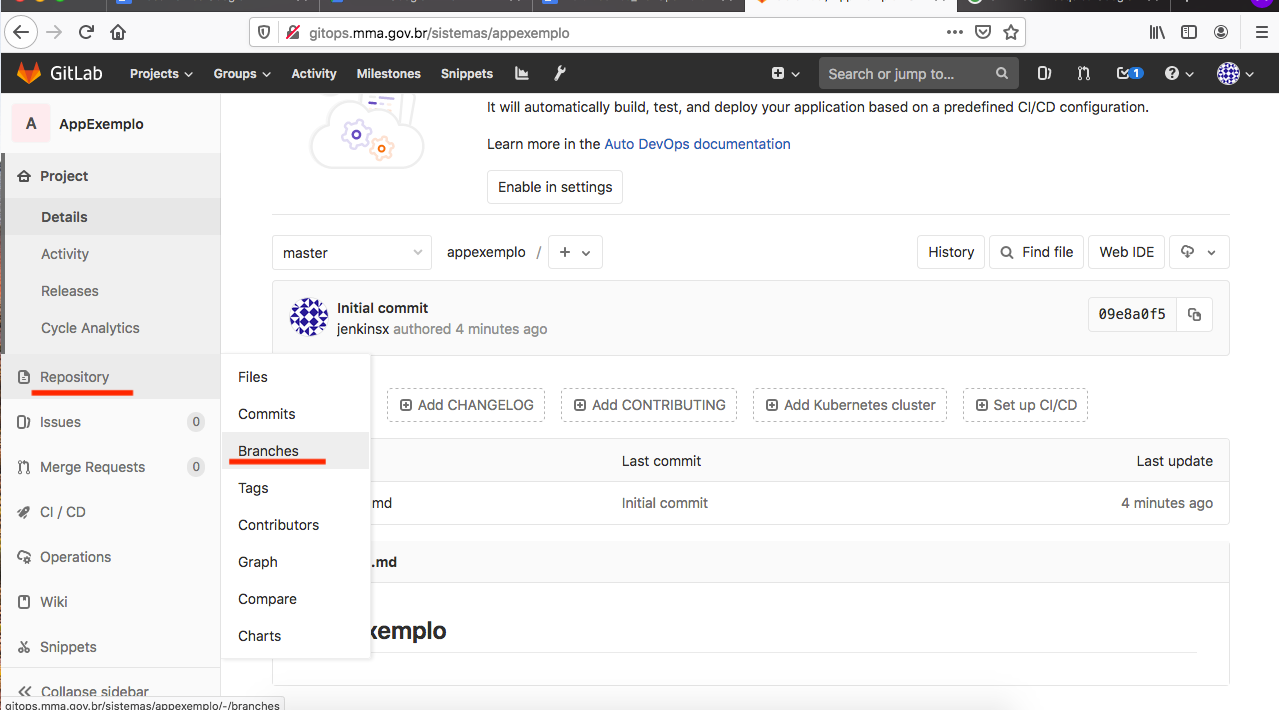
4 - Definir as regras de interação com as branches.

Exemplo: AppExemplo.

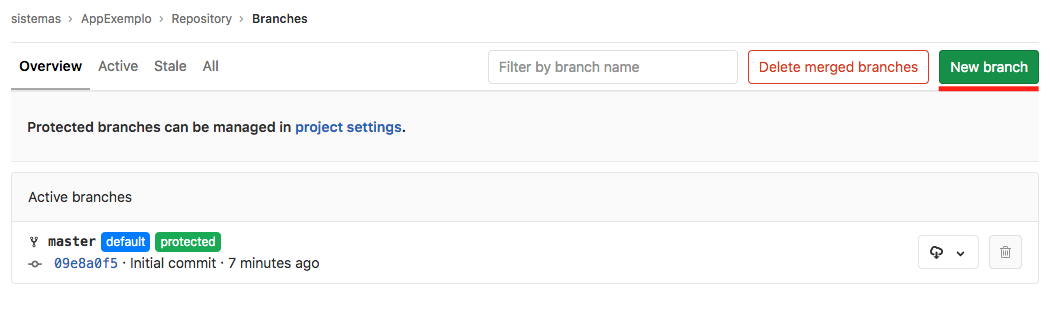


Clique em: Create Project

Com o repositório criado e o arquivo do README, por padrão é criada a branch master. Agora vamos acessar as configurações de branchs para criar as demais: develop e homolog.

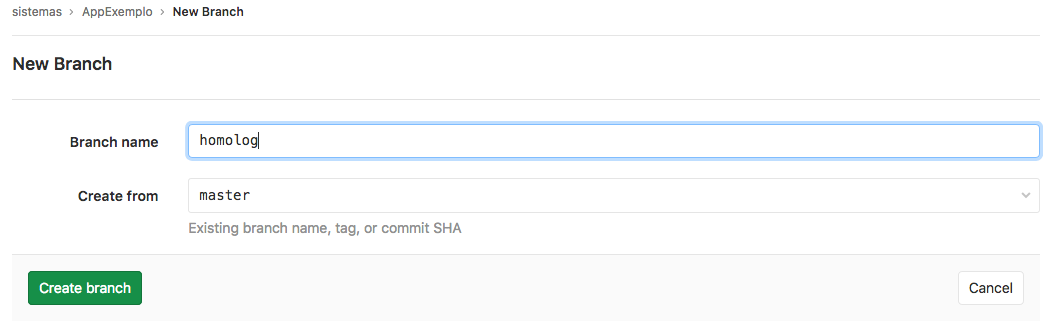


Na página de gestão das branchs

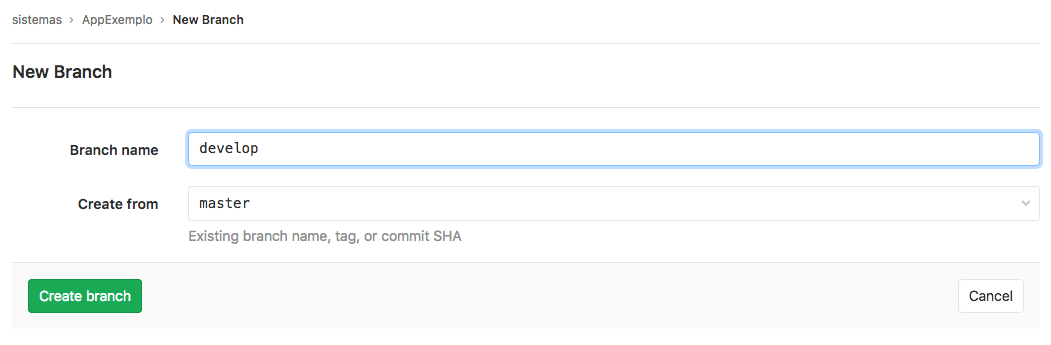


Clique em: New branch

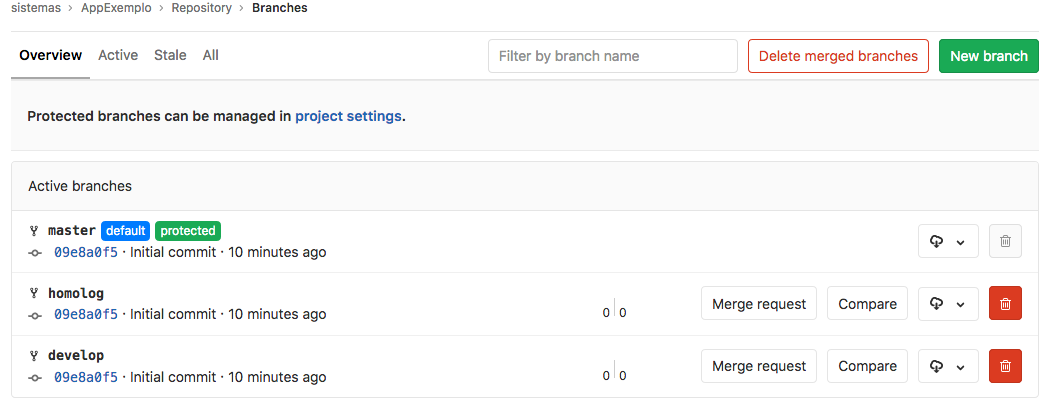
Crie a branch homolog, baseada na master:



Mesmo processo para a develop:

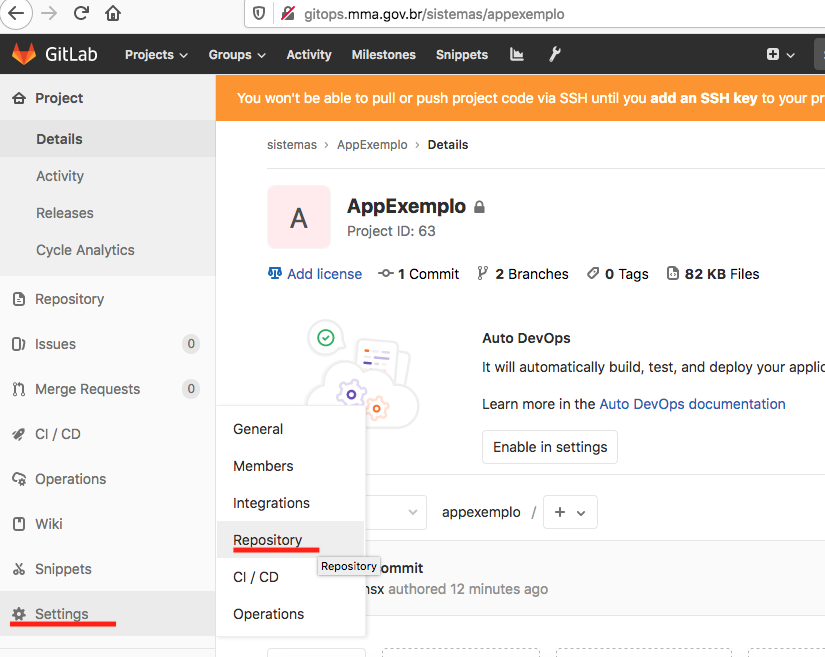


Após criadas temos a seguinte listagem:

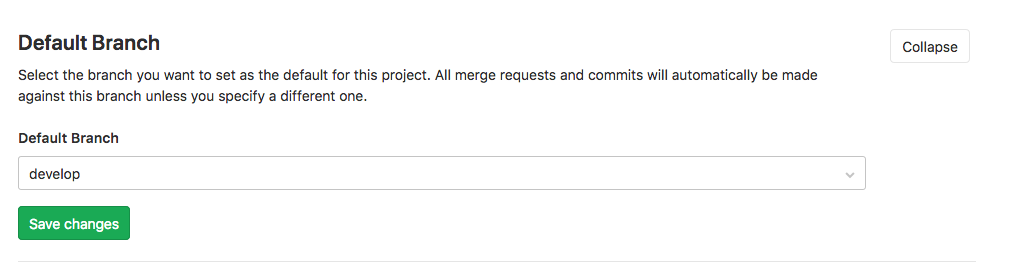


Agora vamos alterar a branch default da master para a develop.

Em Settings -> Repository:



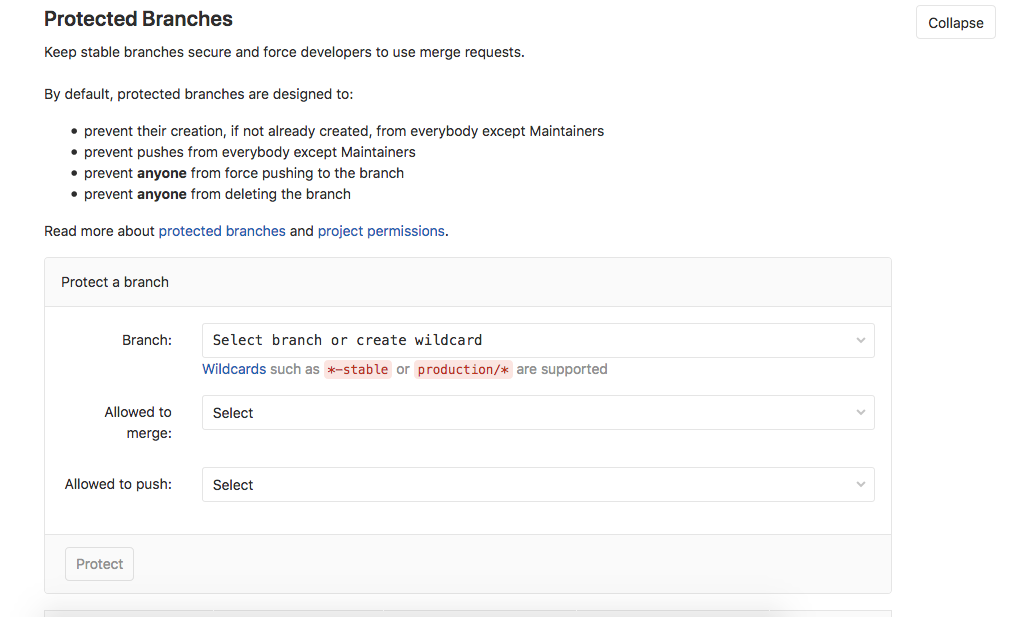
Na seção: Default Branch, expanda as opções e escolha a develop.



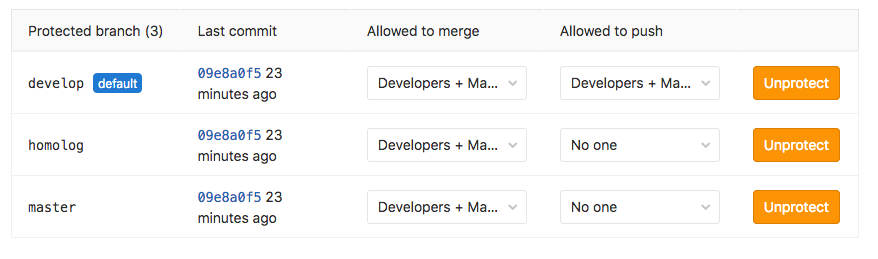
Clique em Save Changes.

Pronto, branch default alterada para develop.

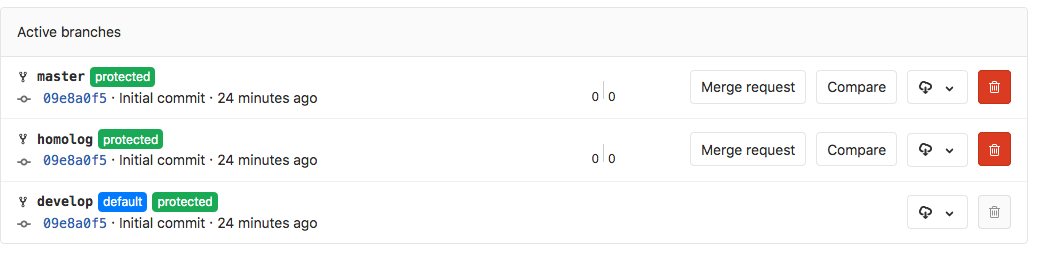
Na mesma página, um pouco mais abaixo, na seção: Protected Branches. Vamos alterar as configurações de acesso nas branches.



Altera para:



Ao final teremos:



# Permissionamento nos repositórios no GITLAB

Com os repositórios criados, os acessos são concedidos para os usuários informados pelas fábricas, via chamado para equipe de infraestrutura. O acesso dentro do Gitlab é feito para cada repositório que o usuário deve ter acesso no perfil: **developer**. Padrão do gitlab

Os acessos para os repositórios de ambiente, devem ser concedidos **apenas** para: dev e hmg.

* **infra/environment-jx-APP-dev.**
* **infra/environment-jx-APP-hmg.**

# Exemplo de Artefatos

**[Atividade deverá ser realizada por ambas as equipes DEV/INFRA]**

Para a execução da esteira contemplar ações de CI e CD é necessário adicionar ao projeto alguns artefatos para o Jenkins X realizar atividades como execução de build, empacotamento da aplicação etc.

|  |  |  |
| --- | --- | --- |
| **Artefato** | **Obrigatório** | **\*Nível de Customização** |
| **Jenkinsfile** | SIM | Geralmente apenas se tem atividade de build |
| **Dockerfile** | SIM | Depende exclusivamente da aplicação |
| **Skaffold** | SIM | Muito Baixo |
| **Helm Chart** | SIM | Médio - Depende das características de infra da app (Portas etc.) |
| **sonar-project.properties** | Não | Muito Baixo |

\***Nível de customização**, baseada nos artefatos disponibilizados de exemplo.

Para facilitar a inclusão de uma nova aplicação na esteira, foi criado um repositório no GITLAB do MMA com arquivos funcionais e um longo README com descrição de cada artefato e o local onde deve ser armazenado em relação ao código.

Repositório: **http://gitops.mma.gov.br/sistemas/artefatos-cicd.git**

A ideia do repositório, além de descrever o papel de cada componente, é servir de modelo para novas aplicações, alterando apenas o necessário, como informando em cada artefatos de exemplo e adicionando ao código da nova aplicação.

É orientado que as equipes de desenvolvimento, tenham acesso READONLY nesse repositório.

Em alguns artefatos, como o skaffold.yml normalmente será necessário alterar apenas o nome do componente que deverá fazer o build da imagem docker.

Para facilitar essas alterações mais simples, foi criado no repositório um script que troca várias referências do nome do componente.

Exemplo:

Nova aplicação: sistema123

$ git clone <http://gitops.mma.gov.br/sistemas/artefatos-cicd.git>

$ cd artefatos-cicd/scripts

$ ./change\_appname.sh sistema123

$ cp skaffold.yaml sonar-project.properties DIRETORIO\_DA\_APP

# Criação de Ambientes no Jenkins X

**[Atividade exclusiva da equipe de infraestrutura]**

Na execução do projeto, foram definidos que cada aplicação terá 3 ambientes.

* **dev** - Desenvolvimento
* **hmg** - Homologação
* **prod** - Produção

Como informado anteriormente o ponto de entrada na esteira é o ambiente de **dev.**

As fábricas de software devem publicar uma versão funcional do código no ambiente de **dev**, para validar além do funcionamento incluído (feature) ou corrigido (bugfix) a integração com componentes da infraestrutura do MMA como AD se for o caso, banco de dados etc.

No Jenkins X, os ambientes são **namespaces** no cluster do OKD e toda a configuração que representa o que está aplicado nesse ambiente é feita usando a filosofia GITOPS, logo, para cada ambiente teremos um repositório no GITLAB.

O CLI do jenkins X fornece a opção de criação de ambientes:

# jx create env

O comando solicita algumas informações como, Estratégia de promoção, Organização dentro do GIT, etc.

**BUG identificado:**

Durante o projeto a execução padrão do comando, apresentava um BUG ao escolher a ORGANIZAÇÃO dentro do GITLAB, no caso para os ambientes do grupo do gitlab: “**infra**”.

Infelizmente, mesmo solicitando o uso do grupo “**infra**”, o repositório era criado abaixo do usuário do gitlab, no caso o “**jenkinsx**”.

Para contornar este bug, durante a criação do ambiente usando o comando: “jx create env”, ao ser perguntado se o Jenkins X deveria criar o repositório, basta informar: Não.

Então é solicitado a URL de um repositório existente.

Exemplo:

# jx create env -n cau-prod -l cau-prod --namespace cau-prod --git-provider-url='http://gitops.mma.gov.br/' --git-private=true --git-provider-kind='gitlab' -p Manual

# Saida completa:

? Domain: apps.mma.gov.br

? *We will now create a Git repository to store your cau-prod environment, ok?* : **No**

? *Git URL for the Environment source code:* **http://gitops.mma.gov.br/infra/environment-jx-cau-prod.git**

? Git branch for the Environment source code: master

INFO[0031] Creating cau-prod Environment in namespace jx

INFO[0031] Created environment cau-prod

INFO[0031] Namespace cau-prod created

Para facilitar essa atividade e configurações do pipeline, foi criado um repositório no GITLAB, para ser usado como código padrão de novos ambientes, incluindo um README detalhando todos os passos para a criação e informações do funcionamento do repositório.

Repo para novos ambientes: **http://gitops.mma.gov.br/infra/environment-jx-base.git**

Ao usar esse repositório como base para novos ambientes, basta realizar pequenas alterações em alguns arquivos, como trocar o nome do namespace. Para facilitar essa atividade, foi criado um script no próprio código, detalhado no README do repositório.

Para visualizar os ambientes criados, acesse o MASTER1:

# jx get envs

Ao final, com os ambientes criado teremos no GITLAB uma estrutura similar:



# Importando aplicações para o Jenkins X

**[Atividade exclusiva da equipe de infraestrutura]**

Depois dos ambientes criados (Dev, Hmg e Prod), como descrito anteriormente e do repositório da aplicação com os artefatos com as devidas alterações, também como descrito anteriormente.

Finalmente importamos a aplicação para o jenkins X. O processo de importar a aplicação no jenkins X consiste em: Registrar essa aplicação nos CRD (Custom Resource Definition) de controle, criar o job no jenkins, criar os webhooks no gitlab, etc.

No Master1:

# oc project jx

# jx import --no-draft=true --disable-updatebot —url URLDOGIT

Opções usadas:

**--no-draft=true** : Por padrão o jenkinsx tenta identificar a linguagem no código da aplicação e criar os artefatos de acordo com o identificado, porém como foram realizados vários ajustes no fluxo, a orientação é usar os artefatos disponibilizados no: <http://gitops.mma.gov.br/sistemas/artefatos-cicd.git> ao invés dos criados automaticamente.

**--disable-updatebot**: Útil para aplicações que usam maven, evita baixar dependências do maven no host onde foi executado o comando jx.

**--url**: URL git do repositório da aplicação.

Exemplo: <http://gitops.mma.gov.br/sistemas/cau-frontend.git>

Assim que importar o projeto, um job criado no jenkins já identifica o Jenkinsfile no repositório da aplicação e inicia a esteira.

# Permissionamento nos ambientes

**[Atividade exclusiva da equipe de infraestrutura]**

Como a ideia é dar autonomia para as fábricas, porém de maneira controlada, podemos atribuir “role” para cada namespace, com roles específicas para os namespaces de produção, mais restritivas e roles menos restritivas para os namespaces de dev e hmg.

O OKD já vem configurado com uma série de roles que contempla perfis bem específicos.

No caso do MMA, estamos usando as seguintes roles:

* **edit:** Para DEV e HMG
* **view:** Para Prod.

A role “**edit**”: Permite ao usuário acessar/criar/deletar praticamente todos os recursos a nível do namespace, como pods, services, secrets, deployment, etc. Exceto alterar os permissionamento dentro do namespace.

A role “**view”:** Permite acesso de leitura a boa parte dos recursos do namespace, como pods, deployments, configmaps, etc. Porém recursos como secrets, que geralmente são usados para armazenar informações sensíveis como senha, a role não permite sequer a visualização do conteúdo.

Para atribuir uma role a um namespace, precisamos o grupo que terá essa role, no caso do MMA, o acesso ao cluster é através dos usuários LDAP com os grupos previamente cadastrados no OKD.

Exemplo: Associar a role **edit** no namespace: **agatha-dev** para o grupo da fábrica **XPTO**:

Em qualquer MASTER:

# oc adm policy add-role-to-group **edit** **GROUP\_XPTO** -n **agatha-dev**

Muito importante informar o namespace com a opção **-n nomedonamespace**, senão for informado será atribuído ao namespace corrente.

Exemplo: Associar a role **view** no namespace: **agatha-prod** para o grupo da fábrica **XPTO**

# oc adm policy add-role-to-group **view** **GROUP\_XPTO** -n **agatha-dev**

# Concedendo acesso ao Jenkins

O jenkins disponibilizado pelo Jenkins X no cluster OKD, veem configurado com o plugin Openshift Oauth, que reaproveita os usuários do cluster do Openshift dentro do Jenkins, com essa configuração, os acessos dentro do jenkins depende das roles que o usuário tem no namespace: **jx.**

Para os usuários das fábricas, os acessos no Jenkins são de apenas leitura, pois todas as configurações de pipeline estão no Jenkinsfile e a esteira é disparada por eventos no GITLAB.

Para conceder ao jenkins, vamos atribuir a role **view** ao grupo da fábrica de software no namespace jx, onde está localizado o jenkins.

Exemplo: Conceder acesso ao jenkins para a fábrica XPTO:

# oc adm policy add-role-to-group **edit** **GROUP\_XPTO** -n **jx**

# Conclusão

A esteira propõe realizar as atividades de CI e CD de maneira automática e controlada, dando mais autonomia para as fábricas e controle pela equipe de infraestrutura, como qualquer solução automatizada, é necessário seguir padrões pré-estabelecidos para o correto funcionamento da esteira. É importante se atentar na primeira interação das fábricas, para auxiliar e esclarecer qualquer dúvida referente aos ajustes necessários nos artefatos que devem ser construídos.