Criando pipelines com o Jenkins

Samuel Maciel Sampaio
@samukasmk
samuel@smk.net.br

Sobre mim

- Samuel Maciel Sampaio
- 29 anos
- Administrador de sistemas a 6 anos
- Desenvolvedor Backend Python a 2 anos
- Entusiasta da cultura de DevOps
- @samukasmk (telegram, face, gmail)
- http://bit.ly/SamuelSampaio

O que é o Jenkins

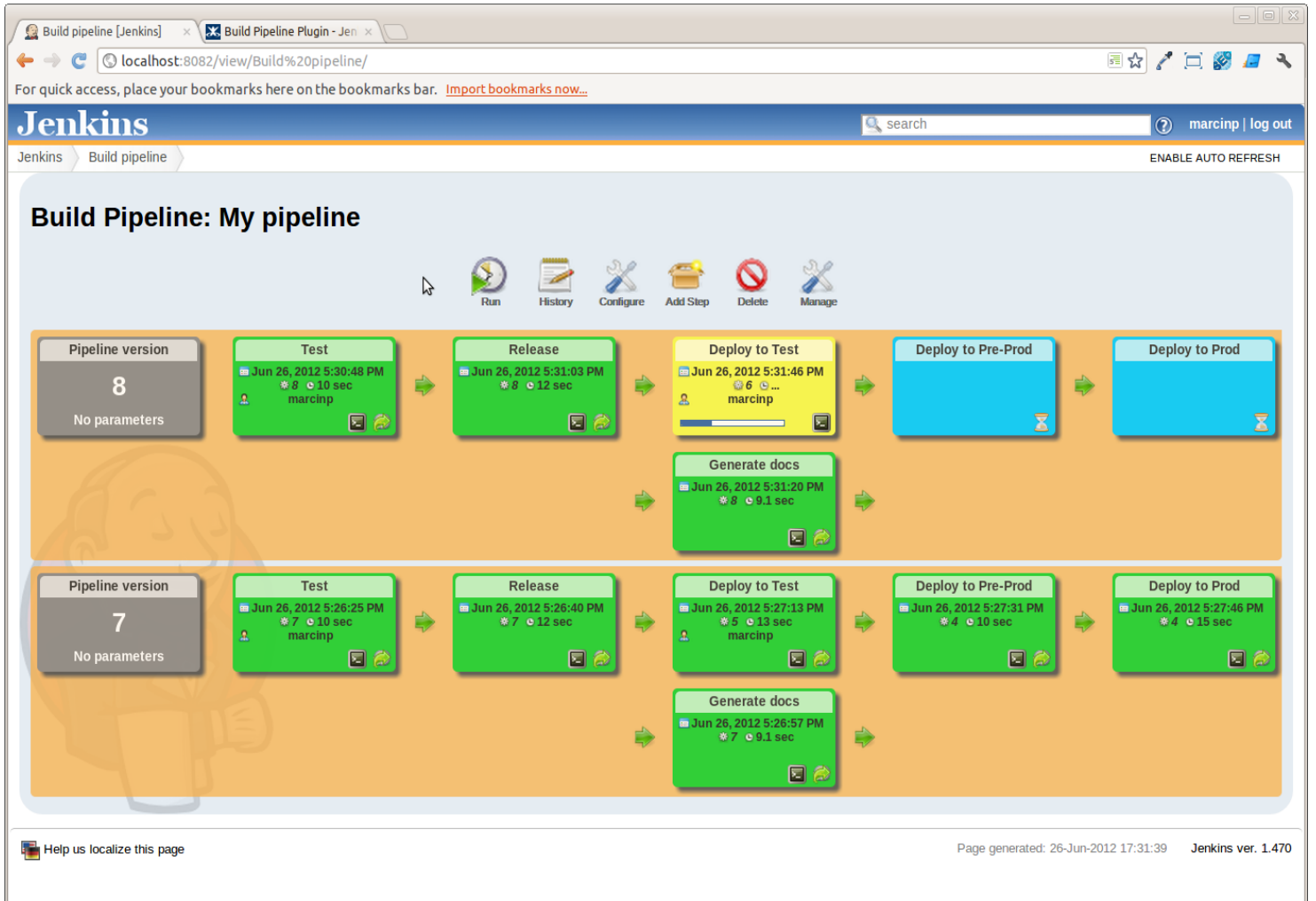
- Jenkins é uma ferramenta de integração e entrega continua.
- Que proporciona mais agilidade e produtividade, na execução de processos rotineiros.
- Tanto de qualidade de software, com a execução automatizada de testes unitários, aceitação e funcional.
- Como em tarefas operacionais de versionamento, deploys e até backups! #OCéuÉOLimite

O que são pipelines

- Pipelines são fluxos de trabalho (workflows), seguimentados por estágios ou passos;
- Agora existe plugins nativos do jenkin para definição de pipelines
- Trazendo mais praticidade ao se defindir fluxos de execução em um unico Job

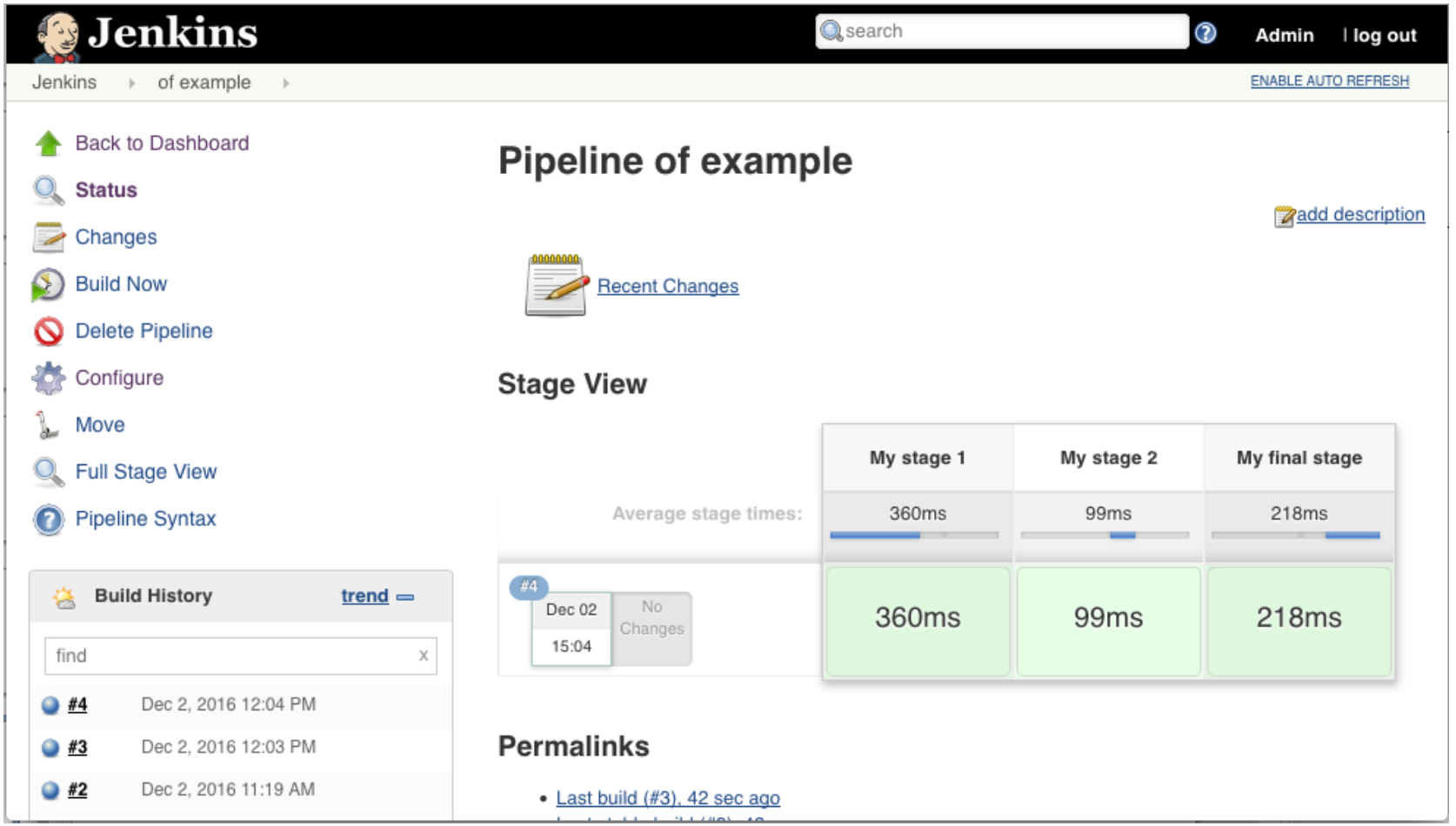
Histórico

- Em meados de 2013 foi lançado o plugin (Build-Pipeline-Plugin) que interconecta varios jobs do jenkins em um workflow



O presente momento

- Em abril de 2016 foi lançado um novo plugin nativo do jenkins, chamado apenas de (Pipeline-Plugin), onde é possivel definir vários passos do mesmo workflow, no mesmo job, em uma visualização de colunas.



Como definir um pipeline customizado

- Os workflows customizados, são definidos por groovy scripts, como uma forma de DSL simples. Onde cada plugin do jenkins disponibiliza os recursos, como funções. Chamados nas documentações de steps
- https://jenkins.io/doc/pipeline/steps/
- Esse groovy script pode ser configurado como string, na configuração do job, ou passado através de um projeto como arquivo Jenkinsfile

Exemplo de um script de pipeline (Jenkinsfile)

```
1  node {
2      stage 'My stage 1'
3      sh 'set'
4
5      stage 'My stage 2'
6      echo "Estou executando no job: " + env.JOB_NAME;
7
8      stage 'My final stage'
9      echo "All is OK!"
10 }
```

Meu case de exemplo

- Desenvolvi uma aplicação em Flask
- Com 3 endpoint:
  - / (home)
  - /ping (exemplo de uma api rest json)
  - /modal (exemplo de um modal html)
- Com testes unitários
  - Se o /ping responde 200, com um json
  - Utilizado (py.test, pytest-flask e pytest-flakes)
- Com testes funcionais
  - Se a pagina externa responde com um modal
  - Utilizado (selenium, pytest-selenium e phantomjs)

https://github.com/samukasmk/grupy-flask-jenkins.git

Executando o projeto na maquina local

- Clonando o projeto do github:
  - git clone https://github.com/samukasmk/grupy-flask-jenkins.git
- Criando o virtual env
  - cd grupy-flask-jenkins
  - virtualenv --python=python3.5 venv
- Instalando as dependencias:
  - ./venv/bin/pip install -r requirements.txt
- Executando a aplicação
  - ./venv/bin/python app.py
- Executando os testes unitários
  - ./venv/bin/py.test -ra -v --flakes tests/unit_tests
- Executando os testes funcionais
  - ./venv/bin/py.test -ra -v --flakes --driver PhantomJS --driver-path=/opt/phantomjs/bin/phantomjs tests/functional_tests

Você percebeu ?
Nós ja temos um pipeline de integração continua !

0. Adicionar gatilho webhook no repositório git
1. Clonar/Efetuar checkout do projeto
2. Criar o virtual env de testes
3. Instalar as dependencias
4. Executar os testes unitários
5. Executar os testes funcionais
6. Disponibilizar para deploy

O tal Jenkinfile...

```
node {
    stage 'Git workspace'
    git 'https://github.com/samukasmk/grupy-flask-jenkins.git'

    stage 'Build venv'
    sh 'virtualenv --python=python3.5 venv'

    stage 'Install deps'
    sh './venv/bin/pip install -r requirements.txt'

    stage 'Unit tests'
    sh './venv/bin/py.test -ra -v --flakes tests/unit_tests'

    stage 'Functional tests'
    sh './venv/bin/py.test -ra -v --flakes --driver PhantomJS ' \
      '--driver-path=/opt/phantomjs/bin/phantomjs tests/functional_tests'

    stage 'Docker push'
    echo 'Se a app estivesse em um container, poderia ser um docker push'
}
```

Hand’s on

Vamos um Job com pipelines no Jenkis

OBRIGADO!

Duvidas?

@SamukaSMk