# Jenkins – Introdução

Jenkins

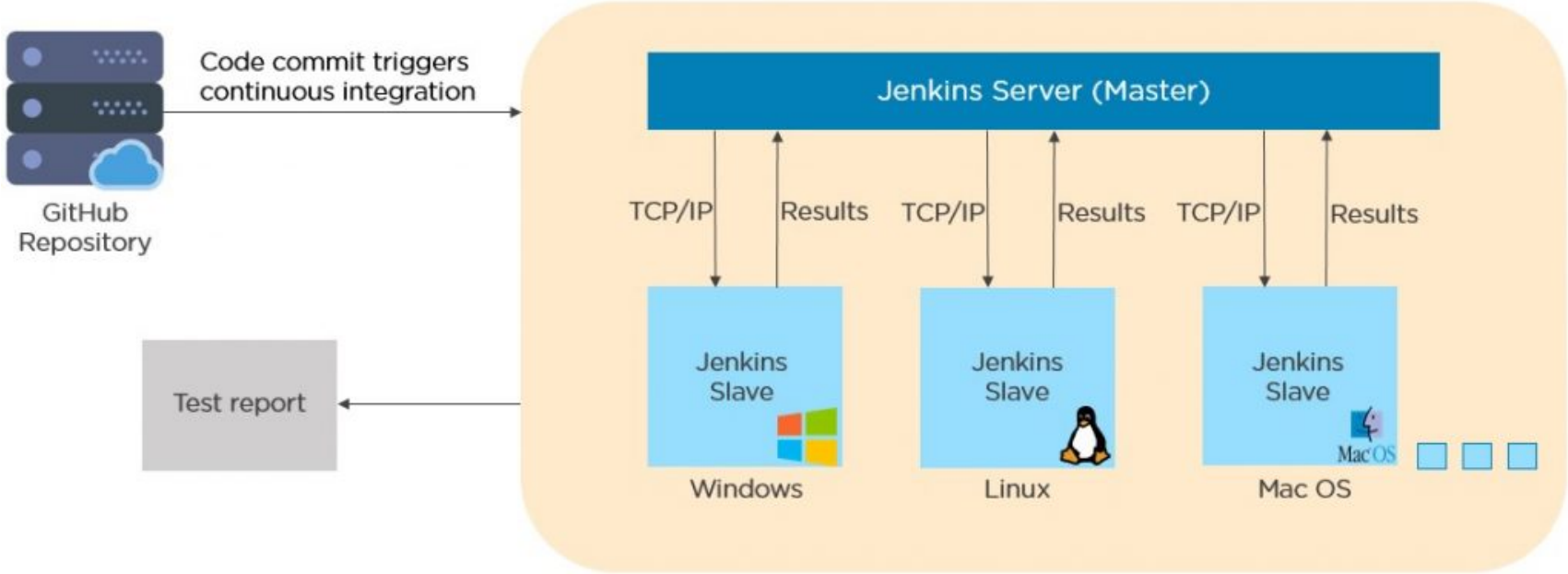
DevOps
Mão na
Massa

# O que é e para que serve?

- Servidor (opensource) de automação escrito em Java
- Amplamente utilizado para CI/CD
- Language Agnostic – Diversos plug-ins para grande parte das linguagens e frameworks.
- Simples Uso – Interface GUI, fácil instalação, suporte a scripts, sem uso de banco de dados, etc.

Jenkins

# Arquitetura



# Jenkins – mão na massa - Instalação

## 1. Vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.hostname = "jenkins"
  config.vm.network "forwarded_port", guest: 8080, host: 8080, host_ip: "127.0.0.1"
  config.vm.provision "shell", path: "provision.sh"
  config.vm.provider "virtualbox" do |v|
   v.memory = 1024
  end
end
```

## 2. provision.sh

```
#!/usr/bin/env bash
echo "Installing Jenkins and dependencies..."
yum install -y java-1.8.0-openjdk
curl --silent --location http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo | sudo tee /etc/yum.repos.d/jenkins.repo
sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
sudo yum install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

**3. Acesso a console:**

http://localhost:8080

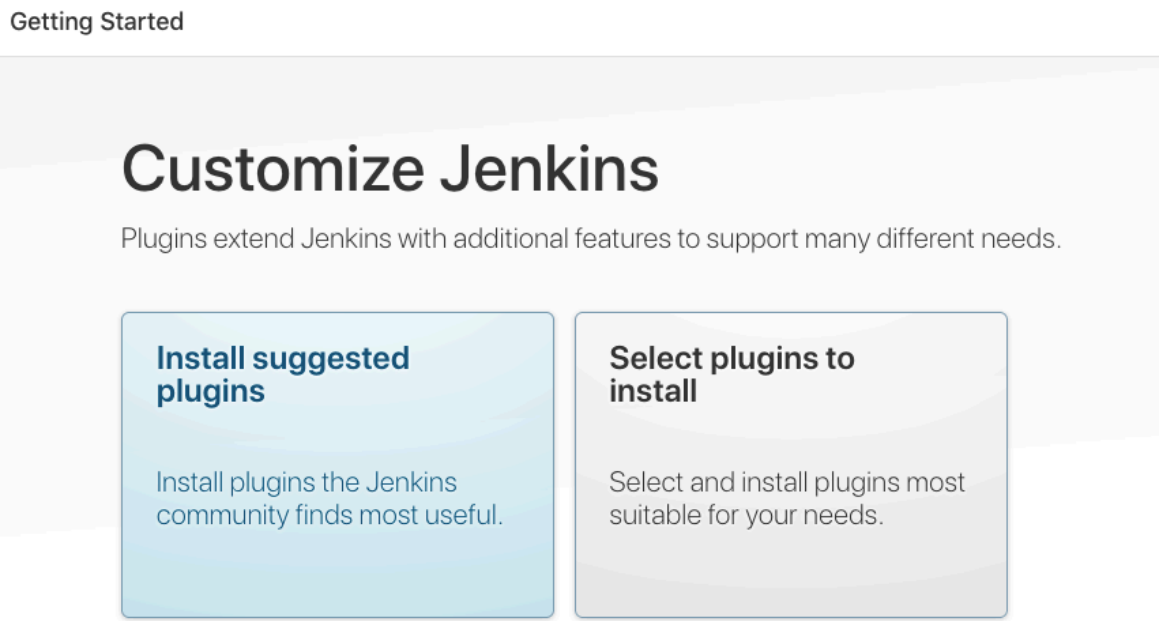
Copiar senha do admin:

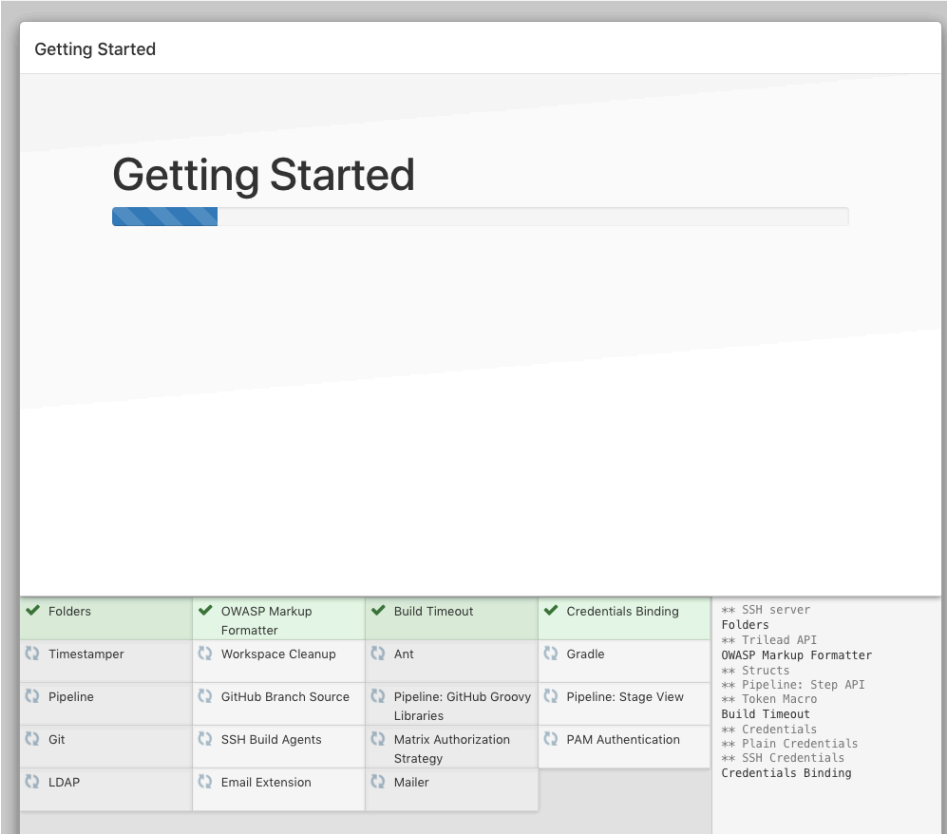
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

4. Plugins:

- GitHub
- Maven

# Jenkins - Instalação padrão de plugins



# Jenkins – mão na massa - Configuração

Getting Started

## Create First Admin User

Username: jenkins

Password: •••••••

Confirm password: •••••••

Full name: jenkins

E-mail address: jenkins@jenkins.com

## Instance Configuration

Jenkins URL: http://localhost:8080/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

# Jenkins – mão na massa – Primeiro Job

**Criar a view: Notes**
**Criar Job:**
Nome: Hello World
Tipo: Pipeline

**Pipeline script:**

```
pipeline {
    agent any
    stages {
        stage("Hello") {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

**Pipeline**: obrigatório para todo inicio do script.
**Agent**: Local onde o script será executado.
**Stages**: Seção que receberá os passos a serem executados pelo job.
**Steps**: Passos executados no pipeline.

# CI / CD - Conceitos

**CI – Continuous Integration**: Toda change criada deverá ser testada a todo momento.

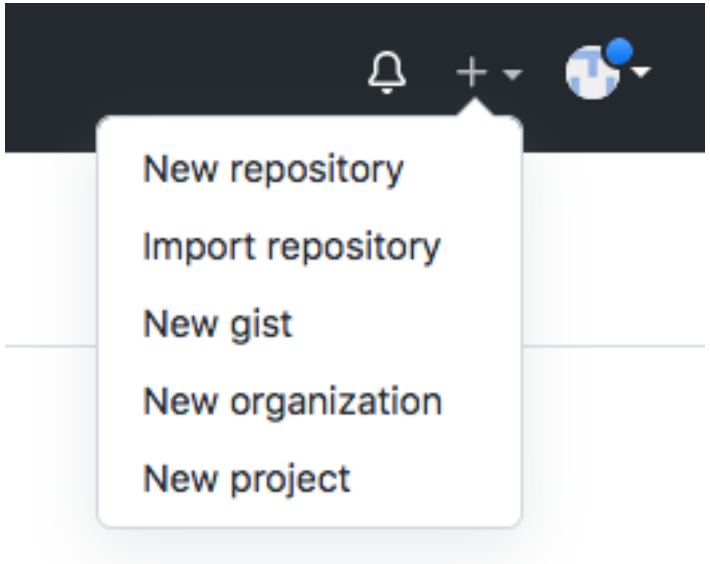
Ex.: cada commit no repositório dispara uma serie de testes (unitários ou de integração) sempre que o código for modificado. Pode conter build (Java para compilação)ou não (como Python).

**CD – Continuous Delivery**: Um passo após o CI. Sempre que o código é testado ele também será disponibilizado (deploy) em um ambiente. Requer passo manual de aprovação humana.

**CD – Continuous Deployment**: Idem ao anterior, mas sem atividade humana.

# Jenkins – CI Pipeline – mão na massa

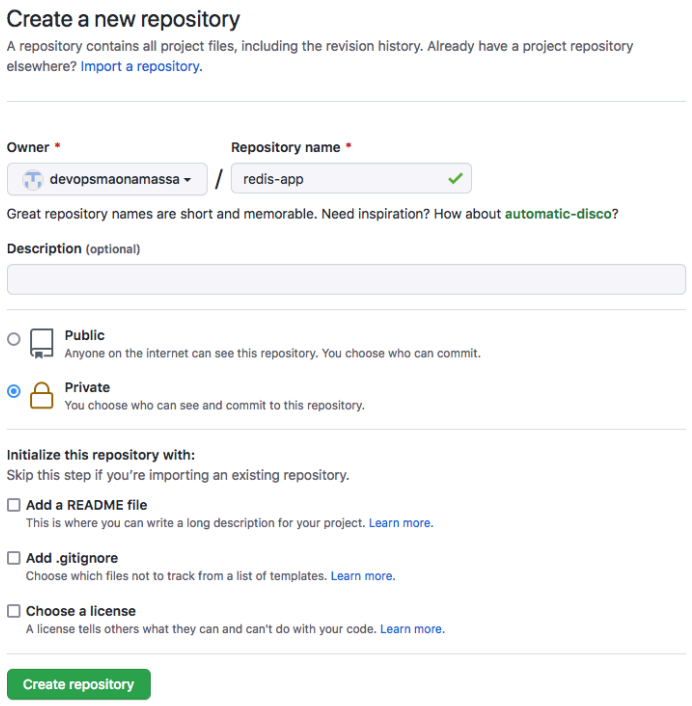
1. Criar novo repo no github – redis-app
2. Criar Jenkinsfile – raiz
3. Criar Dockerfile – raiz
4. Criar docker-compose.yml – raiz
5. Criar teste-integracao.sh – raiz
6. Criar Job no Jenkins



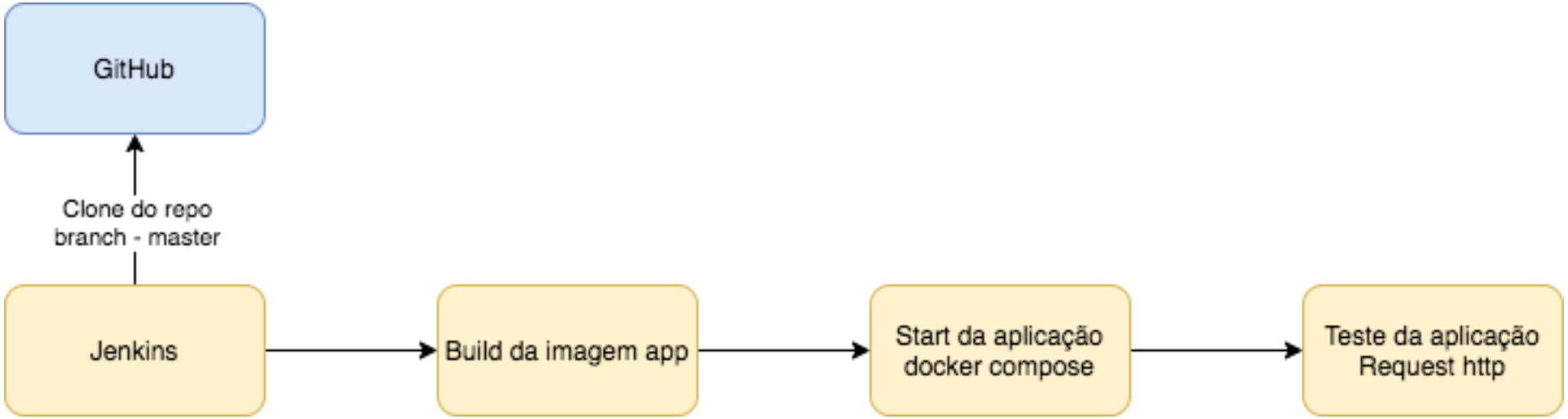

Jenkins

# Jenkins – Criar novo repositório

1. Criar novo repo no github – redis-app
    1. Adicionar como private
2. Clonar o repositório
    1. git clone https://github.com/devopsmaonamassa/redis-app.git
    2. Adicionar Dockerfile da aplicação redis-app lab docker compose
    3. Adicionar docker-compose.yml – lab docker compose



# Jenkins – Pipeline CI



33 lines (31 sloc) | 763 Bytes

```
pipeline {
    agent any
    stages {
        stage('build da imagem docker'){
            steps{
                sh 'docker build -t devops/app .'
            }
        }
        stage('subir docker compose - redis e app'){
            steps {
                sh 'docker-compose up --build -d'
            }
        }
        stage('sleep para subida de containers'){
            steps{
                sh 'sleep 10'
            }
        }
        stage('teste da aplicação'){
            steps{
                sh 'chmod +x teste-app.sh'
                sh './teste-app.sh'
            }
        }
        stage('shutdown dos containers de teste'){
            steps{
                sh 'docker-compose down'

            }
        }

    }
}
```

**Agent**: Servidor que executa o pipeline (master)

**Stage build**: comando docker buil para gerar imagem Docker

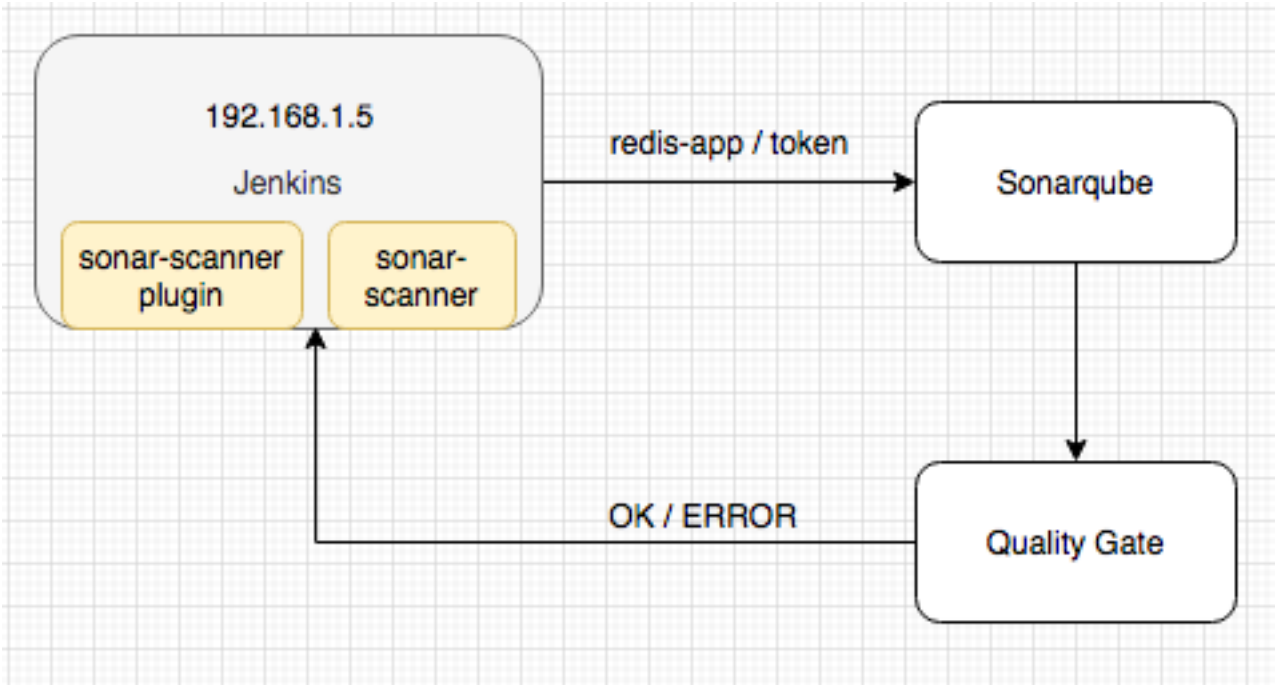
**Stage docker compose**: Subir redis e app via docker compose

**Stage sleep**: sleep de 10 segundo para garantir subida do ambiente

**Stage teste aplicação**: executar teste integrado de chamada http

**Stage shutdown containers**: Baixar containers via docker compose após o teste.

# Integração Jenkins - Sonarqube



# Jenkins – integração com Sonarqube

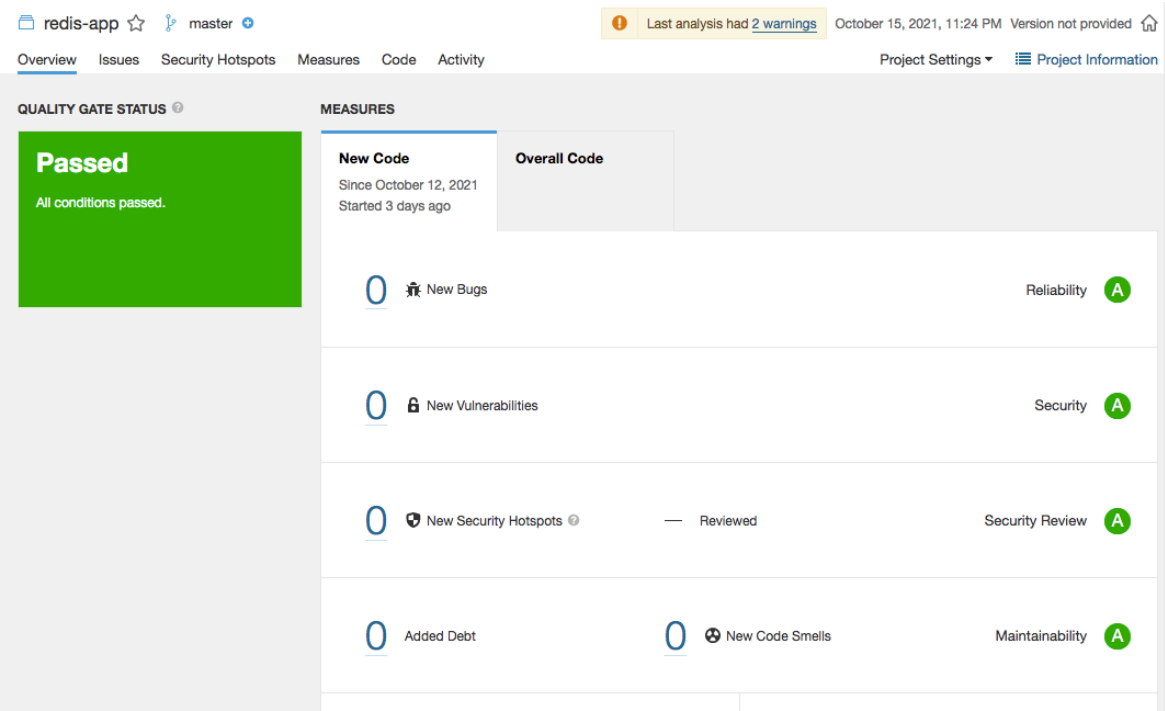
1. Ajustes nos laboratórios:
    1. Configurar private network no Vagrantfile do sonarqube e jenkins
        1. Jenkins: config.vm.network "private_network", ip: "192.168.1.5"
        2. Sonarqube: config.vm.network "private_network", ip: "192.168.1.6”
2. Configs no Jenkins:
    1. Instalar sonar scanner (via provision) – S.O

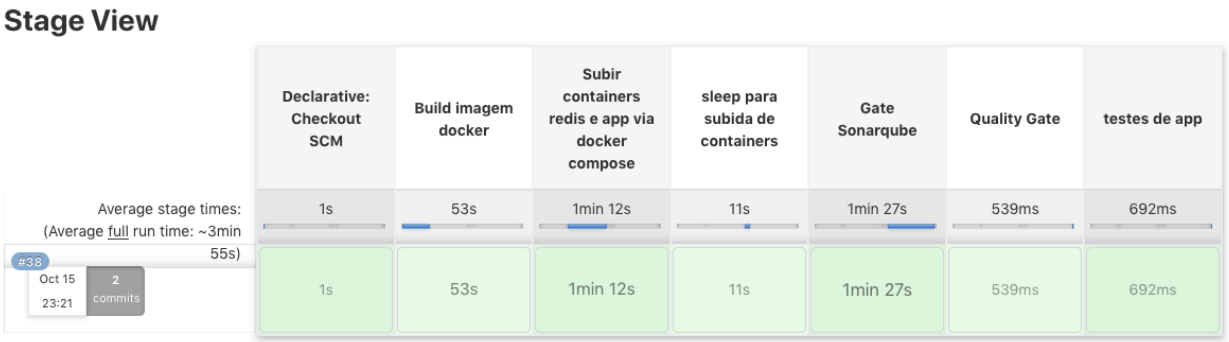
```
yum install unzip -y
wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.6.2.2472-linux.zip
sudo unzip sonar-scanner-cli-4.6.2.2472-linux.zip -d /opt/
mv /opt/sonar-scanner-4.6.2.2472-linux /opt/sonar-scanner
chown -R jenkins:jenkins /opt/sonar-scanner
echo 'export PATH=$PATH:/opt/sonar-scanner/bin' | sudo tee -a /etc/profile
```

    2. Instalar plugin Sonar scanner – Jenkins
    3. Mudar Jenkinsfile e adicionar step de chamada do sonar scanner

3. Configs no Sonar:
    1. Criar profile da aplicação (se nao existir) redis-app
    2. Criar token de acesso – se ainda não existir
    3. Configurar Quality gate Customizado

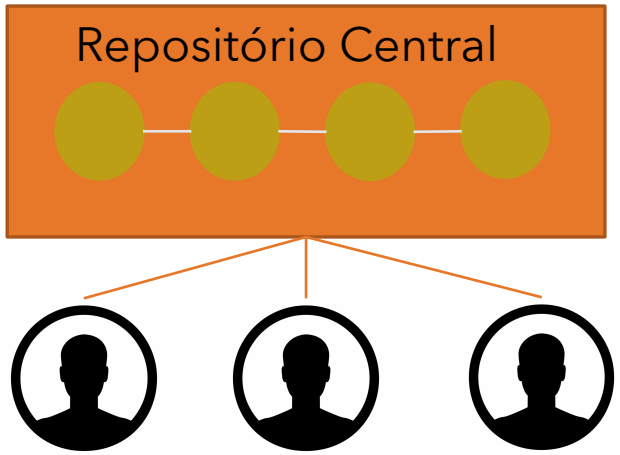
# Pipeline - Sonarqube



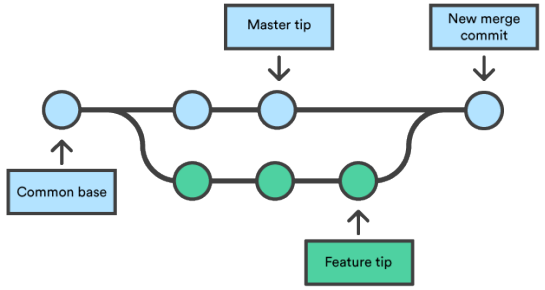



# Git Workflow – estratégia no Jenkins

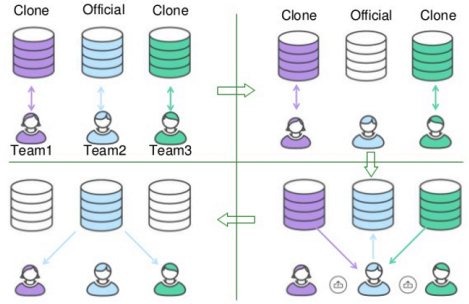
Trunk / Master Workflow




Branch Workflow




Fork Workflow

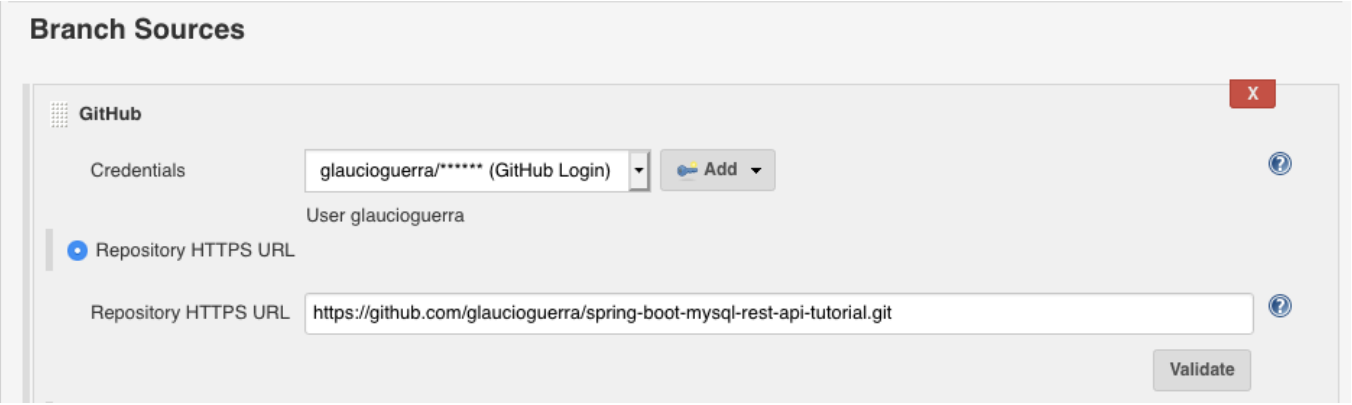



**https://www.atlassian.com/git/tutorials/comparing-workflows**

# Jenkins – Multibranch pipeline – Mão na massa

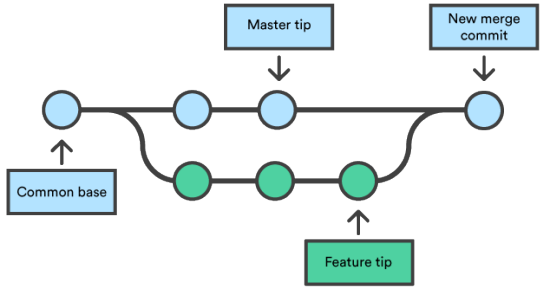
New Item – Multibranch Pipeline
Name: Multibranch-DevOps


**Branch sources**: GitHub
**Repository URL**: Seu repo Git Spring Boot
**Build Configuration**: Mode Jenkinsfile
**Scan Repo triggers**: Interval 1 minute



# Jenkins – Criando novo branch

- git checkout –b nova-funcionalidade
- Adicionar seguinte linha do testes-integracao.sh
  - curl http://localhost:8090/api/notes
- Commit
- Push da branch
- Visualizar job da nova branch
- Efetuar o merge:
  - git checkout master
  - git merge nova-funcionalidade
  - git push origin master




```
macbook:spring-boot-mysql-rest-api-tutorial glaucioguerra$ git merge nova-funcionalidade
Updating d8b8cc0..ae93e90
Fast-forward
 testes-integracao.sh | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

# Jenkins – Job com nova branch

Disable Multibranch Pipeline

Branches (2) | Pull Requests (0)

| S | W | Name ↓ | Last Success | Last Failure | Last Duration | |
|---|---|---|---|---|---|---|
| | | master | 1 day 22 hr - #1 | N/A | 3 min 11 sec | |
| | | nova-funcionalidade | 23 hr - #1 | N/A | 3 min 2 sec | |

# Jenkins Continuous Delivery

Caso de uso:

- Executar Deploy no Swarm
- Subir o manager (se possível mais um nó)
- Obs. 1: Mudar o port foward do Manager se necessário
- Obs. 2: Caso tenha destruído o lab, é necessário iniciar o docker swarm novamente:
  - No manager: **docker swarm init --advertise-addr 192.168.1.2**
  - No worker1: **docker swarm join** --token <TOKEN> 192.168.1.2:2377
- Habilitar acesso via ssh no manager:
  - Editar **/etc/ssh/sshd_config** e remover comentário da linha **PasswordAuthentication yes**
  - Reiniciar serviço ssh: **service sshd restart**

# Jenkins Continuous Delivery

## Configuração no server manager:

- Instalação git e jdk8
- Permissão para usuário vagrant executar docker: **sudo usermod -aG docker ${USER}**

## Configuração no server Jenkins:

- Efetuar login no manager via user jenkins: sudo -u jenkins -g jenkins ssh -v vagrant@192.168.1.2
- Configurar um agente (manager node): Manage Jenkins -> Managed Nodes and Clouds -> New Node
  - Name: swarm-manager
  - Remote root directory: /home/vagrant
  - Usage: Only build Jobs with label expressions matching this node
  - Launch method: Launch agents via ssh
  - Host: 192.168.1.2
  - Credentials: Adicionar nova credential (vagrant /vagrant)
  - Host Key Verification: No verifying Strategy
  - Save

Name: swarm-manager

Description:

# of executors: 1

Remote root directory: /home/vagrant

Labels:

Usage: Only build jobs with label expressions matching this node

Launch method: Launch agents via SSH

Host: 192.168.1.2

Credentials: vagrant/******

Add

Host Key Verification Strategy: Non verifying Verification Strategy

Advanced...

Availability: Keep this agent online as much as possible

# Jenkins Continuous Delivery

- Criar novo docker compose para deploy:

```
docker-compose-stg.yml > Ansible > services > {} app > [ ] ports
1   version: '3'
2   services:
3     mariadb:
4       image: "mariadb:latest"
5       hostname: mariadb
6       environment:
7         MYSQL_ROOT_PASSWORD: "devopsmaonamassa"
8         MYSQL_DATABASE: "notes"
9       ports:
10        - "3306:3306"
11      volumes:
12        - /root/docker/mariadb/datadir:/var/lib/mysql
13    app:
14      image: "glaucio/devopsmaonamassa-app:${TAG}"
15      depends_on:
16      - mariadb
17      ports:
18        - "8080:8080"
```

- Adicionado novo stage no Jenkinsfile para deploy no Swarm:

```
61          stage("Iniciar serviço no docker swarm"){
62              agent {label 'swarm-manager'}
63              steps{
64                  sh "TAG=$BUILD_NUMBER docker stack deploy -c docker-compose-stg.yml app"
65              }
66          }
67      }
68  }
```

# Jenkins Continuous Delivery

- Validar se o deploy ocorreu corretamente:

```
[vagrant@manager ~]$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                               PORTS
fhhstfhx8hbe        app_app             replicated          1/1                 glaucio/devopsmaonamassa-app:14     *:8080->8080/tcp
z3we2a4q53bd        app_mariadb         replicated          1/1                 mariadb:latest                      *:3306->3306/tcp
[vagrant@manager ~]$
```

- Escalar a aplicação para nó worker1:
  - docker service scale app_app=2

```
[root@manager ~]# docker service scale app_app=2
app_app scaled to 2
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
[root@manager ~]#
```

- Validar container no worker1:

```
[root@worker1 ~]# docker ps
CONTAINER ID        IMAGE                               COMMAND                  CREATED
69cf4700c7a8        glaucio/devopsmaonamassa-app:14     "java -jar /easy-not…"   7 seconds ago
[root@worker1 ~]#
```