jenkins&&docker安装

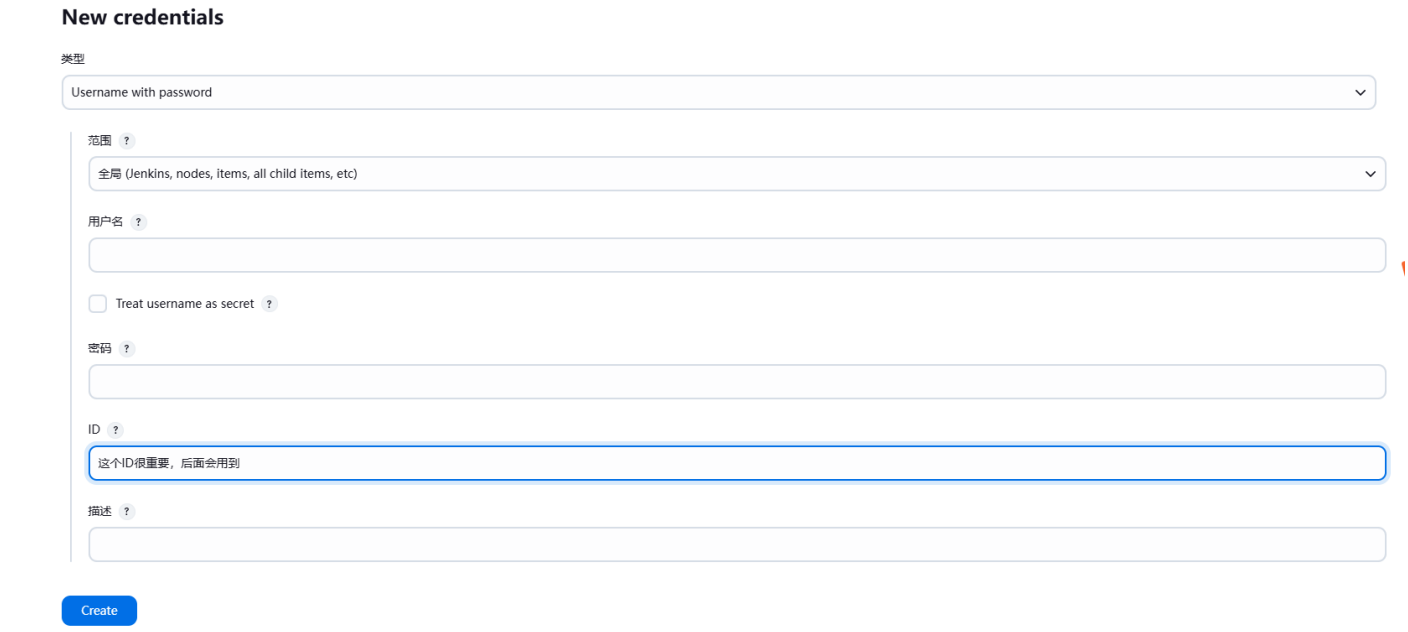
核心点是3个主要挂载的，sock和root和jdk需要根据情况定，

```
version: '3'
services:
  jenkins:
    image: harbor.iovhm.com/hub/jenkins/jenkins:2.488-jdk17
    container_name: jenkins
    restart: always
    privileged: true
    user: root
    ports:
      - "8080:8080"
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - ./m2:/root/.m2
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/bin/docker
    environment:
      - TZ=Asia/Shanghai
```

准备工作

需要先建立拉取代码和推送docker镜像使用的用户名密码凭据

系统管理 > 凭据管理 > 全局 > Add Credentials



New credentials

类型: Username with password

范围: 全局 (Jenkins, nodes, items, all child items, etc)

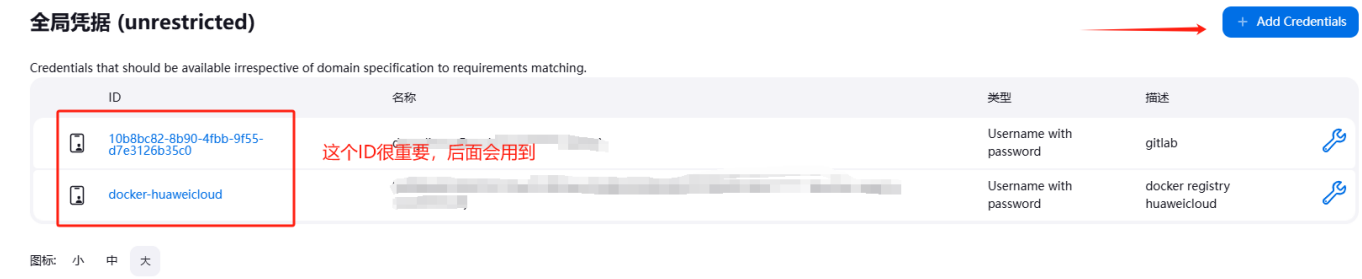
用户名:

☐ Treat username as secret

密码:

ID: 这个ID很重要，后面会用到

描述:



全局凭据 (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	名称	类型	描述
10b8bc82-8b90-4fbb-9f55-d7e3126b35c0	[redacted]	Username with password	gitlab
docker-huaweicloud	[redacted]	Username with password	docker registry huaweicloud

这个ID很重要，后面会用到

新建pipeline （流水线项目）

然后根据项目类型，将后面章节的pipeline脚本复制到项目配置

springboot □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□sh□□□□□□□□□□□□□□□□□□□□(''')□□□□□□□□□□□□□□□□□□□□□□□□□□("""),□□□□□□□□□□□□□sh□□□□□□□□□□□□

- **□□□□□□□□□□sh□□**
- **□□□□□□□sh□□□($)□□□□□(\${var})**

```
pipeline {
    agent any

    // □□□□□□□□□
    environment {
        // GIT□□□
        GIT_URL         = 'https://g.vpclub.cn/park/park-huahai/admin-java.git'
        // GIT □□□
        GIT_IDENTITY    = 'git-szdev-donglietao'
```

```
        // 项目名称，将生成docker镜像的名称的一部分
        PROJECT_NAME    ="gzxfzd"
        // docker仓库的分组名称
        PROJECT_NS      ="vp-park"
        // docker仓库地址
        DOCKER_REGISTRY ="swr.cn-south-1.myhuaweicloud.com"
        // 这三个参数将组合成你的最终docker镜像名称为
        // swr.cn-south-1.myhuaweicloud.com/vp-whdev/springboot-demo
        // DOCKER 仓库凭证
        DOCKER_IDENTITY = 'docker-huaweicloud'

    }

    // 构建过程参数
    parameters {

        // 代码分支
        choice(name: 'GIT_BRANCH', choices: ['development', 'release'],description: '代码分支测试dev,生产release')
        // 镜像TAG，将最被添加到docker镜像标签上
        // 这个参数将被组合到镜像的最终名称
        // swr.cn-south-1.myhuaweicloud.com/vp-whdev/springboot-demo:test
        choice(name:"IMAGE_TAG",choices:["test","prod"],description:"镜像的tag")

        choice(
            name: 'PUSH_SERVICE',
            choices: ['ALL',
'jfast-service/jfast-auth',
'jfast-service/jfast-basic',
'jfast-service/jfast-cloud-docking',
'jfast-devops-center',
'jfast-gateway',
'jfast-oa-services',
'jfast-service/jfast-research-center'
                  ],
            description: '需要推送的服务
       )
    }

    tools {
```

```
        // 这个地方与系统设置的mvn工具包对应
        maven "M3"
        // JDK版本，与全局jdk工具对应
        jdk "JDK-8"
    }

    stages {
        stage('Build') {
            steps {

                // 注意credentialsId参数，填入你前面创建的git仓库用户名IM
                // 注意url参数，填入你的git仓库地址
                git branch: "${GIT_BRANCH}", credentialsId: "${GIT_IDENTITY}", url: "${GIT_URL}"

                // 构建脚本默认在代码仓库根目录运行
                // 如果你的pom.xml文件不在根目录，需要使用dir指令进入到对应的目录
                // 或者使用mvn clean compile package -f ./source/pom.xml 指定pom文件位置
                // 如果你的pom文件是在源代码的根目录，可以删除dir代码块包裹

                dir("./"){
                    // 注意这里的多行脚本引号
                    // 如果是三个单引号，则无法引用局部变量
                    // 如果是三个双引号，可以引用局部变量，但是需要对特殊字符进行转义
                    sh """

                    pwd
                    java -version
                    mvn clean compile package

                    """
                }
            }
        }

        stage("docker"){
            steps{
                script{
                    def SERVICE_MAP=[
"jfast-service/jfast-auth":"jfast-auth",
```

```
"jfast-service/jfast-basic":"jfast-basic",
"jfast-service/jfast-cloud-docking":"jfast-cloud-docking",
"jfast-devops-center":"jfast-devops-center",
"jfast-gateway":"jfast-gateway",
"jfast-oa-services":"jfast-oa-services",
"jfast-service/jfast-research-center":"jfast-research-center"
                    ]

                    // 使用docker凭据,credentialsId为你前面创建的docker仓库凭据
                    withCredentials([usernamePassword(credentialsId: "${DOCKER_IDENTITY}",
passwordVariable: 'docker_password', usernameVariable: 'docker_username')]) {
                        def conditionVal= "${PUSH_SERVICE}"
                        def branch = "${GIT_BRANCH}"
                        if(conditionVal=="ALL"){
                            for(item in SERVICE_MAP.entrySet()){
                                def serviceName = "${item.value}"
                                def serverPath = "${item.key}"
                                dir("./${serverPath}"){
                                    // 注意这里的多行脚本引号
                                    // 如果是三个单引号，则无法引用局部变量
                                    // 如果是三个双引号，可以引用局部变量，但是需要对特殊字符进行转义
                                    sh """

                                    pwd
                                    echo ${serviceName}
                                    docker login -u ${docker_username} -p ${docker_password}
${DOCKER_REGISTRY}
                                    docker build -t
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}/${serviceName}:${IMAGE_TAG}-${BUILD_NUMBER} .
                                    docker push
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}/${serviceName}:${IMAGE_TAG}-${BUILD_NUMBER}


                                    """
                                }

                            }
                        }else{
                            def serviceName =SERVICE_MAP[conditionVal]
                            def serverPath = conditionVal
```

```
                            dir("./"){
                                // 注意这里的多行脚本引号
                                // 如果是三个单引号，则无法引用局部变量
                                // 如果是三个双引号，可以引用局部变量，但是需要对特殊字符进行转义
                                sh """

                                pwd
                                echo ${serviceName}
                                docker login -u ${docker_username} -p ${docker_password}
${DOCKER_REGISTRY}
                                docker build -t
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}/${serviceName}:${IMAGE_TAG}-${BUILD_NUMBER} .
                                docker push
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}/${serviceName}:${IMAGE_TAG}-${BUILD_NUMBER}

                                """
                            }
                        }
                    }
                }

        }
    }
}
```

springboot maven Pipeline

```
pipeline {
    agent any

    // 环境变量，全局可用
    environment {

        GIT_URL         = 'https://g.vpclub.cn/park/crabapple-drink/crabapple-drink.git'
        // GIT 仓库凭证
        GIT_IDENTITY    = 'git-szdev-donglietao'
```

```
        // 项目名称，将生成docker镜像的名称的一部分
        PROJECT_NAME    = 'crabapple-drink'

        // docker仓库的分组名称
        PROJECT_NS      = 'vp-park'
        // docker仓库地址
        DOCKER_REGISTRY = 'swr.cn-south-1.myhuaweicloud.com'
        // 这三个参数将组合成你的最终docker镜像名称为
        // swr.cn-south-1.myhuaweicloud.com/vp-whdev/springboot-demo
        // DOCKER 仓库凭证
        DOCKER_IDENTITY = 'docker-huaweicloud'



    }

    // 构建过程参数
    parameters {

        // 代码分支
        choice(name:"GIT_BRANCH",choices:["development","release"],description:"代码分支')
        // 镜像TAG，将最被添加到docker镜像标签上
        // 这个参数将被组合到镜像的最终名称
        // swr.cn-south-1.myhuaweicloud.com/vp-park/springboot-demo:test
        choice(name:"IMAGE_TAG",choices:["test","prod"],description:"镜像的tag")
    }

    tools {

        // 这个地方与系统设置的mvn工具包对应
        maven "M3"
        // jdk版本，与全局jdk工具对应
        // jdk "JDK-8"
    }

    stages {
        stage('Build') {
            steps {

                // Get some code from a gitlab
                // 注意credentialsId参数，填入你前面创建的git仓库用户名IM
```

```
                // 注意url参数，填入你的git仓库地址
                git branch: "${GIT_BRANCH}", credentialsId: "${GIT_IDENTITY}", url:
"${GIT_URL}"

                // Run Maven on a Unix agent.
                // 构建脚本默认在代码仓库根目录运行
                // 如果你的pom.xml文件不在根目录，需要使用dir指令进入到对应的目录
                // 或者使用mvn clean compile package -f ./source/pom.xml 指定pom文件位置
                // 如果你的pom文件是在源代码的根目录，可以删除dir代码块包裹

                dir("./renren-cloud-tenant/"){
                    sh """

                    echo \$(pwd)
                    mvn clean compile package

                    """
                }
            }
        }

        stage("docker"){
            steps{
                // docker build 默认在命令行运行的根目录查找Dockerfile
                // 如果你的Dockerfile不在根目录，需要使用dir指令进入到对应的目录
                // 如果你的Dockerfile在根目录下，可以删除dir代码块包裹
                dir('./renren-cloud-tenant/park-admin/park-admin-server/') {

                    // 使用docker凭据，credentialsId为你前面创建的docker仓库凭据
                    withCredentials([usernamePassword(credentialsId: "${DOCKER_IDENTITY}",
passwordVariable: 'docker_password', usernameVariable: 'docker_username')]) {

                        sh """

                        echo \$(pwd)
                        docker login -u ${docker_username} -p ${docker_password}
${DOCKER_REGISTRY}
                        docker build -t
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}:${IMAGE_TAG}-${BUILD_NUMBER} .
                        docker push
```

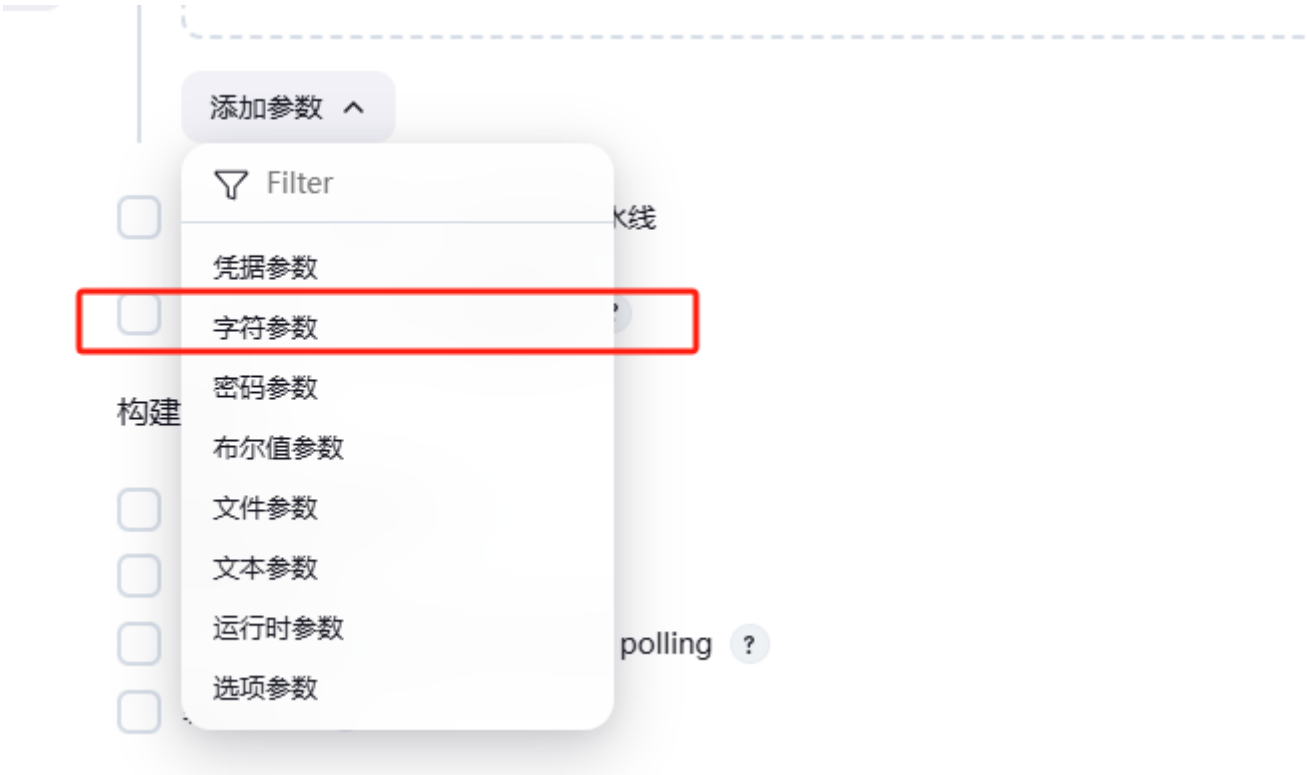
```
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}:${IMAGE_TAG}-${BUILD_NUMBER}



                        """
                    }
                }
            }
        }
    }
}
```

点击运行的时候没有出现构造参数界面

在项目配置选项里面，勾选参数化，然后与Pipeline脚本中设置的环境变量对应上。



nodejs Pipeline

```
pipeline {
    agent any

    // 环境变量，全局可用
    environment {
        // GIT仓库地址
        GIT_URL          = "https://g.vpclub.cn/park/sdyk/front-web-admin.git"
        // GIT 仓库凭证
        GIT_IDENTITY     = "donglietao-vpclub-sz-git"
        // 项目名称，将生成docker镜像的名称的一部分
        PROJECT_NAME     ="nodejs-demo"
        // docker仓库的分组名称
        PROJECT_NS       ="vp-whdev"
        // docker仓库地址
        DOCKER_REGISTRY ="swr.cn-south-1.myhuaweicloud.com"
        // 这三个参数将组合成你的最终docker镜像名称为
        // swr.cn-south-1.myhuaweicloud.com/vp-whdev/nodejs-demo
        // DOCKER 仓库凭证
        DOCKER_IDENTITY = "docker-huaweicloud"
        // nodejs专用，nodejs内存溢出错误时候需要
        NODE_OPTIONS     = "--max-old-space-size=4096"

    }
```

```
// 构建过程参数
parameters {
    // 代码分支
    choice(name:"BRANCH",choices:["development","release"],description:"代码分支")
    // 镜像TAG，将被添加到docker镜像标签上
    choice(name:"IMAGE_TAG",choices:["test","prod"],description:"镜像的tag")
    // 这个参数将被组合到镜像的最终名称
    // swr.cn-south-1.myhuaweicloud.com/vp-whdev/nodejs-demo:test
}


stages {
    stage('Build') {
        steps {
            // Get some code from a gitlab
            // 注意credentialsId参数，填入你前面创建的git仓库用户名密码
            // 注意url参数，填入你的git仓库地址
            git branch: "${BRANCH}", credentialsId: "${GIT_IDENTITY}", url: "${GIT_URL}"

            // build
            // 需要使用的nodejs版本，与全局设置的nodejs工具包名称对应
             nodejs('node-18.20') {
                sh """

                echo \$(pwd)
                node -v
                npm -v

                # 非必须，有时候npm install卡主
                # npm cache clean --force
                # rm -rf node_modules
                # rm package-lock.json
                # rm yarn.lock

                npm add yarn --registry=https://registry.npmmirror.com

                ./node_modules/.bin/yarn -v
                ./node_modules/.bin/yarn install --registry https://registry.npmmirror.com
```

```
                    # 如果不使用yarn
                    # npm install --registry=https://registry.npmmirror.com

                    if   [ "${IMAGE_TAG}" = "prod" ]; then
                      ./node_modules/.bin/yarn run build:prod
                    elif [ "${IMAGE_TAG}" = "test" ]; then
                      ./node_modules/.bin/yarn run build:test
                    else
                        echo "IMAGE_TAG is neither 'prod' nor 'test'. No build script
executed."
                    fi

                    """
                }
            }
        }

        stage("docker"){
            steps{

                // docker build 默认在命令行运行的根目录查找Dockerfile
                // 如果你的Dockerfile不在根目录，需要使用dir 指令进入到对应的目录
                // 如果你的Dockerfile在根目录下，可以删除dir 代码块包裹

                withCredentials([usernamePassword(credentialsId: "${DOCKER_IDENTITY}",
passwordVariable: 'docker_password', usernameVariable: 'docker_username')]) {

                    sh """

                    echo \$(pwd)
                    docker login -u ${docker_username} -p ${docker_password}
${DOCKER_REGISTRY}
                    docker build -t
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}:${IMAGE_TAG}-${BUILD_NUMBER} .
                    docker push
${DOCKER_REGISTRY}/${PROJECT_NS}/${PROJECT_NAME}:${IMAGE_TAG}-${BUILD_NUMBER}

                    """

                }
```

```
            }
        }
    }
}
```

nodejs第二种方法，多阶段构建

由于某些依赖需要比较复杂的环境，例如gyp，早期的sass，需要很多依赖，导致整个jenkins环境配置异常复杂，这个时候可以使用docker的多阶段构建

```
FROM harbor.iovhm.com/hub/node:14.21.3 AS build
WORKDIR /data
COPY ./ /data
RUN cd /data && \
    yarn install --registry=https://registry.npmmirror.com && \
    yarn run build:prod && \
    true

FROM harbor.iovhm.com/hub/nginx:1.14.2
COPY --from=build /data/dist/ /usr/share/nginx/html/management
```

- **AS build**的类似与给这个阶段取了个名字
- 注**--from=build**，可以实现内容共享

相对于第一种方法，则只需要获取代码，打包全部有docker完成

版本号 #55
由 董列涛 创建于 24 九月 2024 03:06:54
由 董列涛 更新于 8 八月 2026 09:18:00