**Introduction for Jenkins server**

Server:

Server 1: Jenkins server, 10.239.147.197 (Jenkins 所在的server: /var/lib/jenkins)

Server 2: Images server, 10.239.147.186 (存放编译成功后的images: /home/acrn/images)

UserName: root/acrn

PWD: acrn@123

其中 server 1 只有一块500G的硬盘. Server 2 有一块256G的SSD, 一块2T的硬盘, 一块1T的硬盘, 4块500G的硬盘 (组成了一虚拟硬盘). 系统安装在了SSD中. 其他硬盘分别挂载在/home/acrn/images/中的pr, post, engineer中.

安装

sudo apt-get install jenkins openjdk-8-jdk

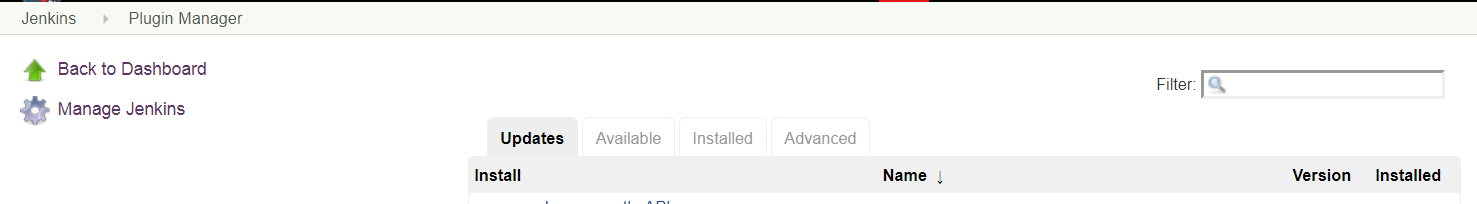
登录

登入 <http://localhost:8080>

第一次登入的时候,是administrator用户. 初始密码在/var/lib/jenkins/secrets/initialAdminPassword 文件中.

安装plugin  
 这里列出一些plugin.   
 Ant Plugin; Apache HttpComponents Client 4.x API Plugin; built-on-column; Conditional BuildStep; Credentials; Git client; Git plugin; GIT server Plugin GitHub API Plugin; GitHub Branch Source Plugin; GitHub Integration; GitHub plugin; GitHub PR Comment Build Plugin; GitHub Pull Request Builder; Javadoc Plugin; JDK Tool Plugin; Multijob; Multiple SCMs plugin; Parameterized Remote Trigger; Parameterized Trigger plugin; Rebuilder; Run Condition; SCM API Plugin; SSH Credentials Plugin; SSH Slaves; 等等…

后期可以在Plugin Manager中安装



Update: 是指有哪些已经安装的插件需要update

Available: 是可以安装的插件. 如果需要安装新的插件, 可以在这里安装.

Installed: 已经安装的插件

Advanced: 配置Jenkins的网络代理; 手动load 插件; 等

Jenkins 导航

如下图所示:

New Item: 创建一个新的Jenkins Job

People: 对用户进行管理

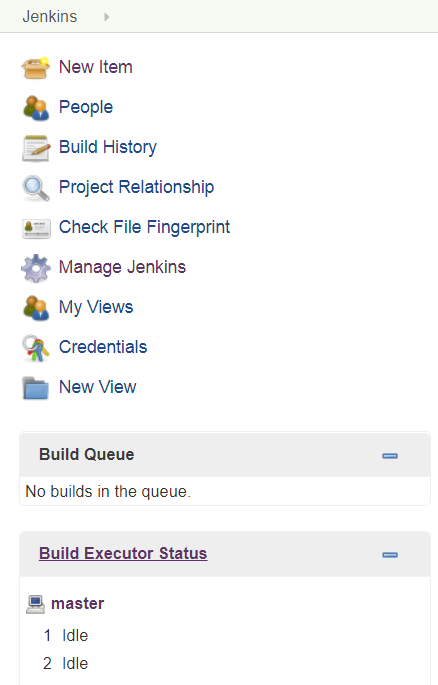
Manage Jenkins: 配置Jenkins的信息

My Views: 列出所有的Jenkins Job

Credentials: 这里列出所有的Jenkins中保存的验证书. 用来连接node或者GitHub 验证

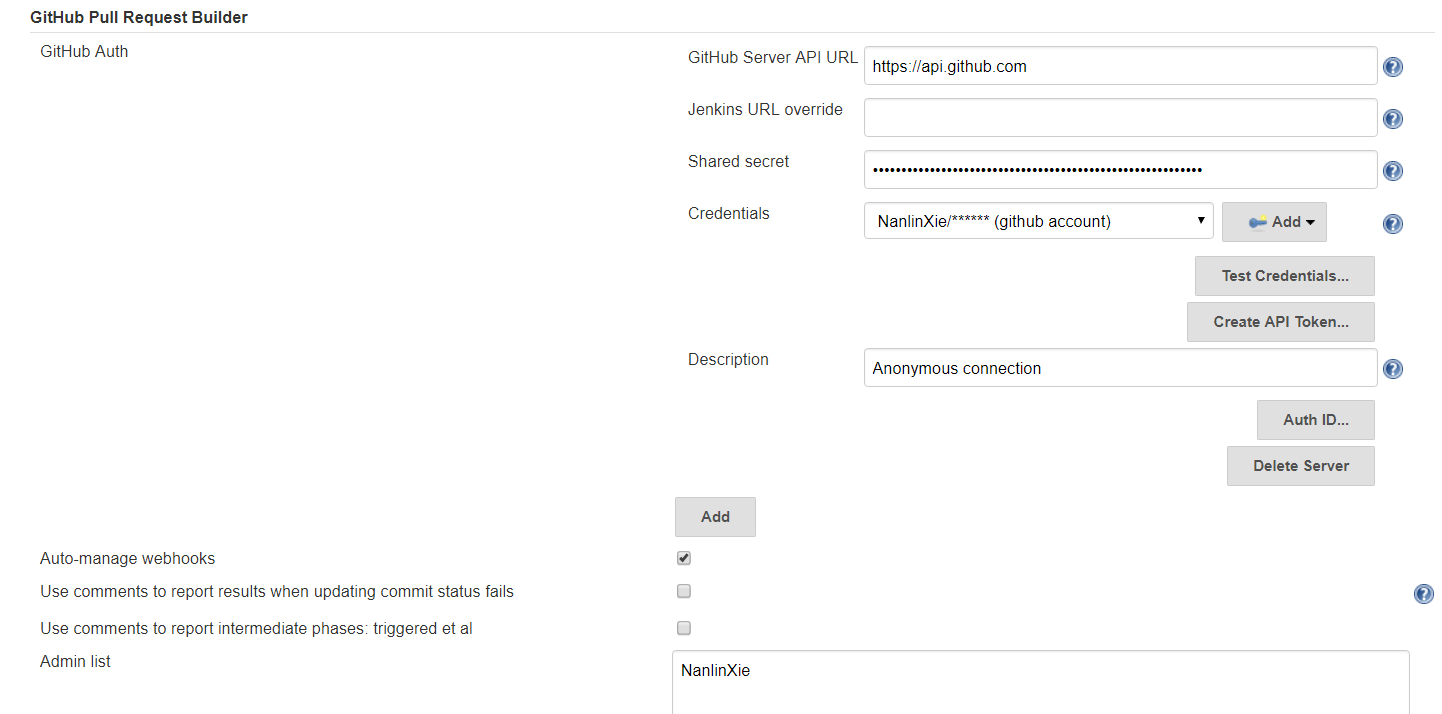
New View: 创建 view. 用来区分不同的Jenkins Job

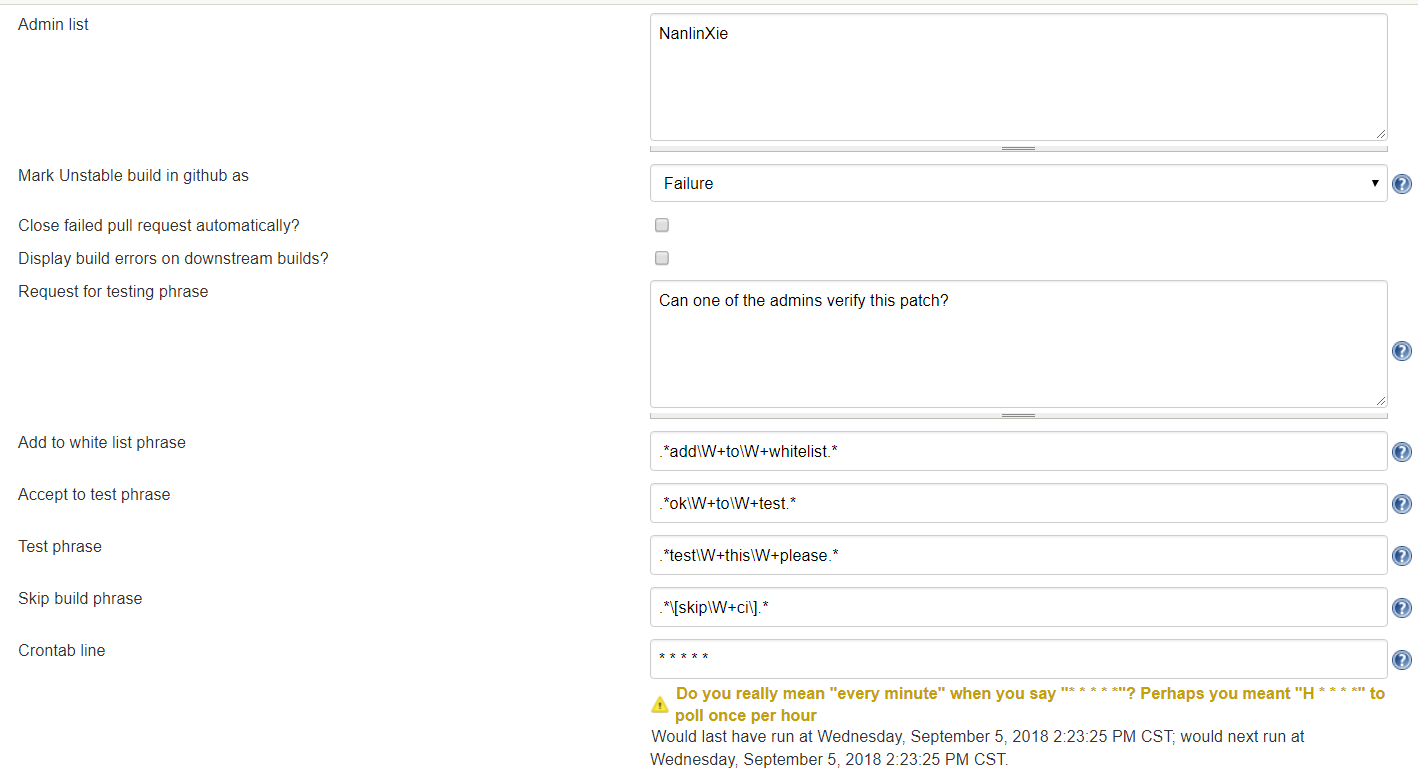
Build Executor Status: 列出所有Node, 和提供create node的接口.



Manage Jenkins -> System configuration:

这里只介绍一下GitHub Pull Request Builder





GitHub Server API URL: <https://api.github.com>

Shared secret: 1f9fb349006ef0b8b64a42869080401d97d06c14

Credentials: NanlinXie/xxxxxx

Auto-manage webhooks: 勾选

Admin list: NanlinXie

Make Unstable build in GitHub as: Failure

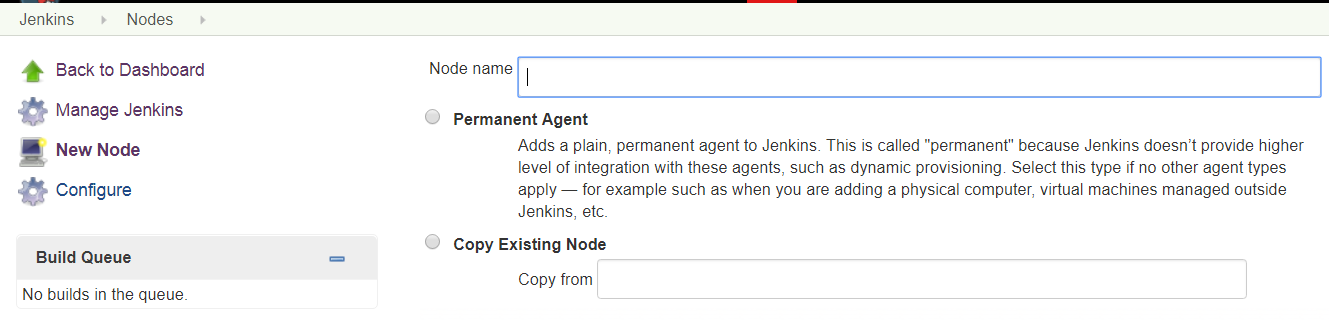
Request for testing phrase: Can one of the admins verify this patch.

Crontab line: \*\*\*\*\* (mean every minute)

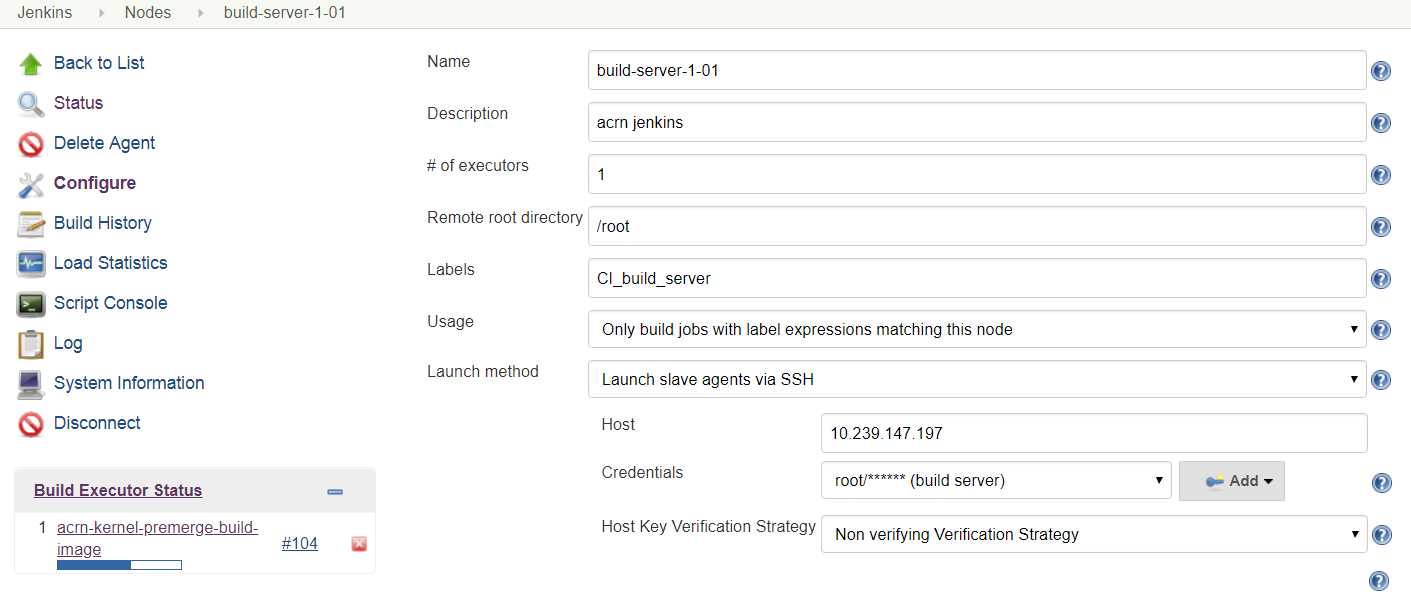
Create Node

点击Build Executor Status. 跳转到Node的配置页面. 页面地址是: <http://acrn-jenkins.sh.intel.com:8080/computer/>

接下来点击New Node. 创建Node有两种方式, 一是**Copy Existing Node**, 及创建的Node是copy一个已经存在的Node. 一是**Permanent Agent,** 创建一个Node. 需要自己去配置具体的内容.



填写 Node name. 注意不要与已经存在的同名, 和创建方式. 点击OK 按钮.



上图中是创建Node的详细信息.   
 Name: 为node 的name.

Remote root directory: 是Node的工作的主目录. 我们需要在这个目录里创建一个workspace的文件夹.

Labels: 不同的Node可以是同一个Labels, 可看成是组成一个group.

Usage: 选择 only build jobs with label expressions matching this node.

Launch method: 选择Launch slave agents via SSH; 及使用ssh连接node.

Host: 提供node的IP地址

Credentials: 登录node的username和pwd

Host Key Verification Strategy: 选择Non verifying Verification Strategy.

Create Jenkins Job

点击New item-> 输入item的name ->选择创建Item的类型. 我们一般是选择 Freestyle project 或者MultiJob Project->OK

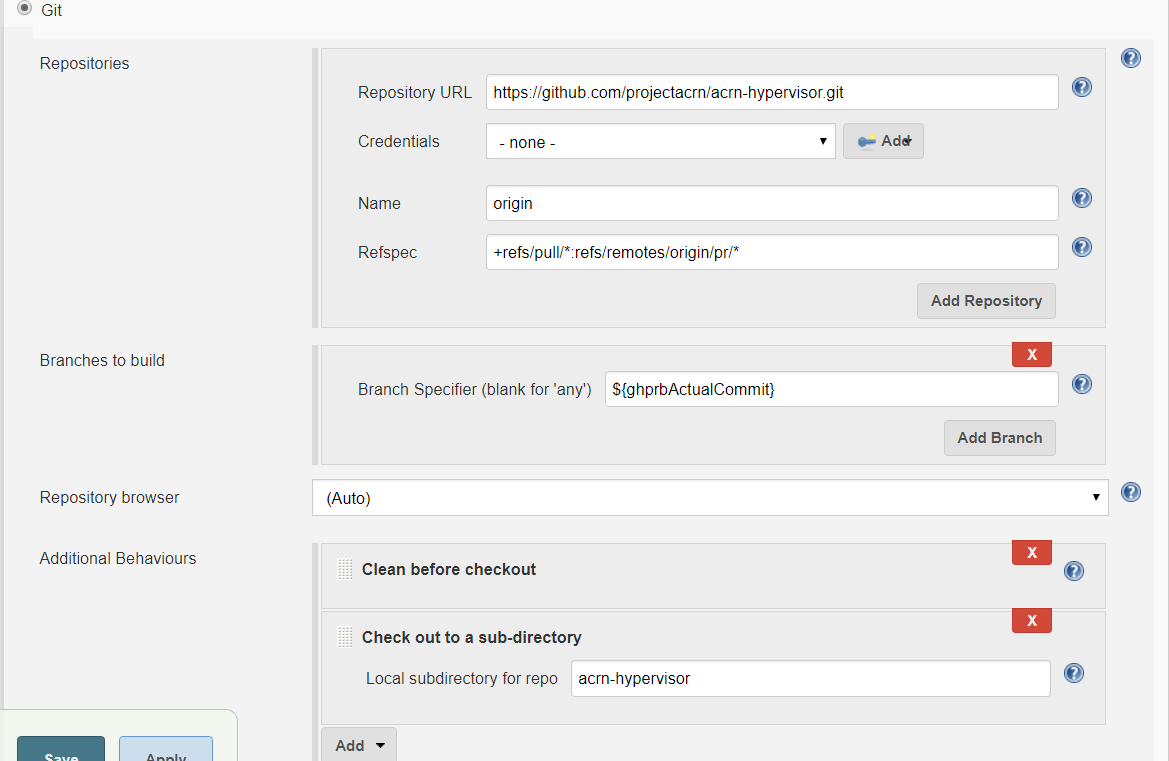
这里我们以acrn-hypervisor-premerge-build-image为例:

配置GitHub Project option: Project url: 该Item 支持的github project的地址,如<https://github.com/projectacrn/acrn-hypervisor>

配置This project is parameterized: 点击Add Parameter选择需要的parameter的类型. 这里是需要一个 String parameter, parameter 的name是ghprbActualCommit. 该参数是GitHub Pull Request Builder 插件需要使用.是要GitHub的PR trigger该job时传入的参数.

配置Restrict where this project can be run: 该Item 是在哪个Label 或者 Node上运行.

配置 Source Code Management:



Repository URL: <https://github.com/projectacrn/acrn-hypervisor.git> 下载code的地址.

Name: origin, 远程库中的唯一ID

Refspec: +refs/pull/\*:refs/remotes/origin/pr/\* 下载的代码详细配置. 是否需要该PR

Branches to build: code的branch

Additional Behaviours: Check out to a sub-directory(保存到本地的文件夹.) Advanced clone behaviours(下载代码过程的一些配置. 重要的是一个 timeout)

配置 Build Triggers:

GitHub Pull Request Builder: 及该item 由PR Trigger.

GitHub API credentials: <https://api.github.com>: Anonymous connection

Admin list: NanlinXie

Trigger phrase: OK to verify (再次Trigger的标志)

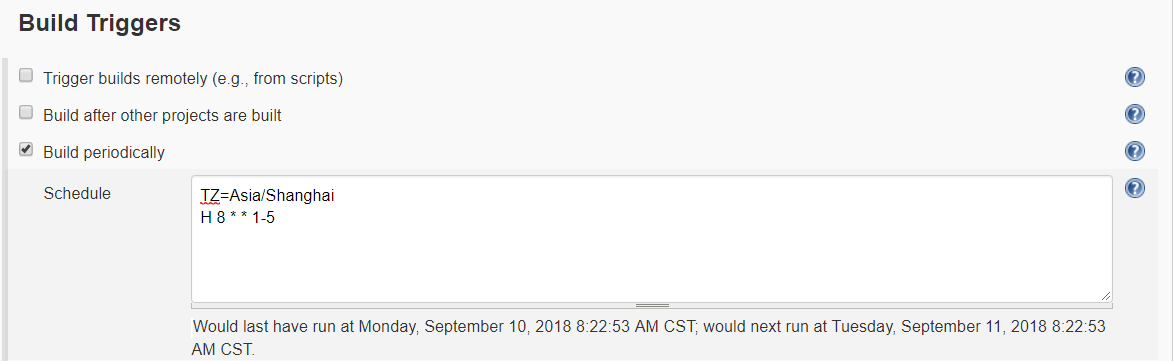
Skip build phrase: .\*\[skip\W+ci\].\* ( skip trigger 的标志)

Crontab line: \*\*\*\*\* (every minutes)

White list: 白名单

Allow members of whitelisted organizations as admins: enable

这里还介绍一种自动trigger的例子



Build periodically: 按时间去trigger job.

TZ表示时区, H8\*\*1-5表示星期1到星期5, 早上8点运行.

配置Build:

Execute shell: 执行bash 脚本. 这里有全局变量 JOB\_NAME(Item的名字), BUILD\_NUMBER(Item被触发时的number), ghprbPullId(GitHub PR Builder提供的变量, 及PR的ID), ghprbPullLink(PR的link)

MultiJob Phase: 如下图

Phase name: 该Phase的名字

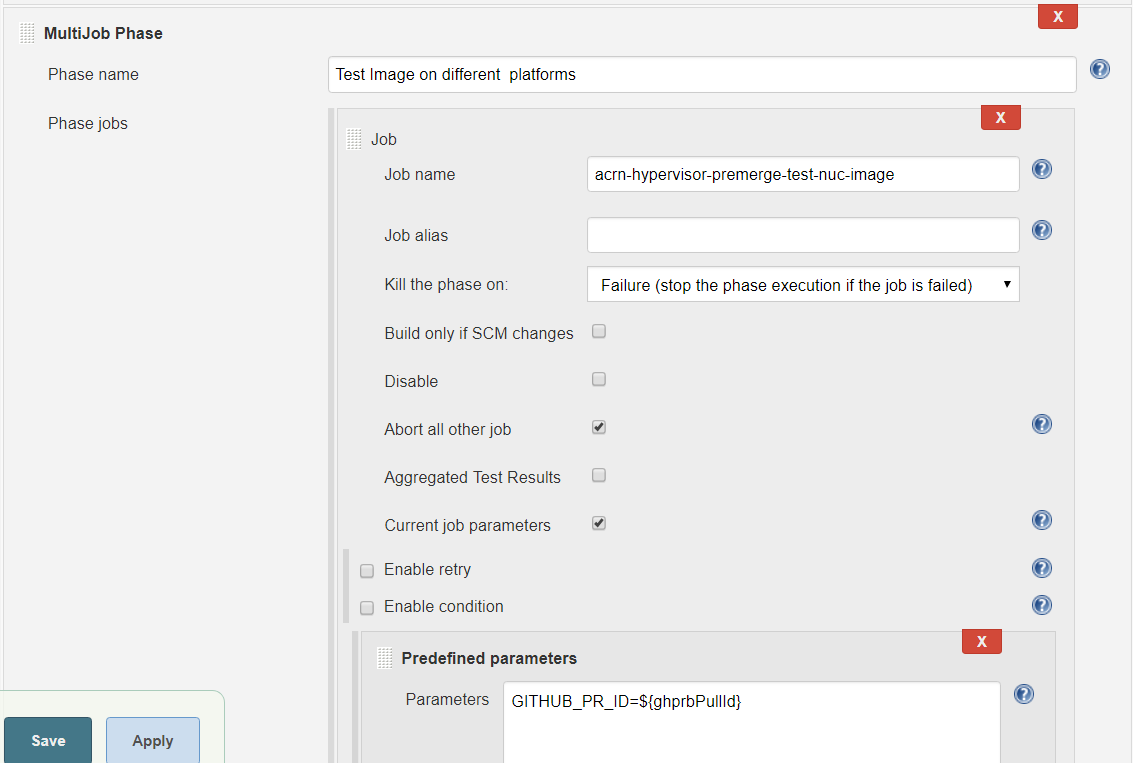
Phase jobs: job name: 子job的名字.

Kill the phase on: 什么样的状况退出程序.

Abort all other job: 失败后是否不在执行其他子job

Current job parameters: 是否使用当前job的Parameter.

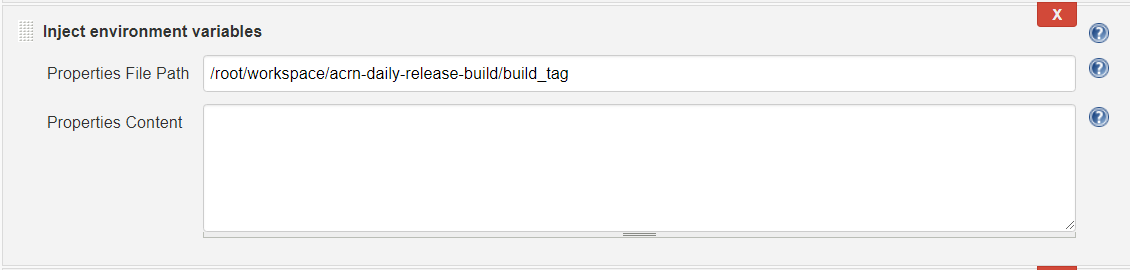
Predefined Parameter: 即传到子job的参数. 该参数在子job里使用.



使用插件添加jenkins 内置变量的方法.

我们使用build 的Inject environment variables 操作项去添加Jenkins 的内置变量

如下图:



我们把要添加的变量放到一个文件中.

如: cat build\_tag

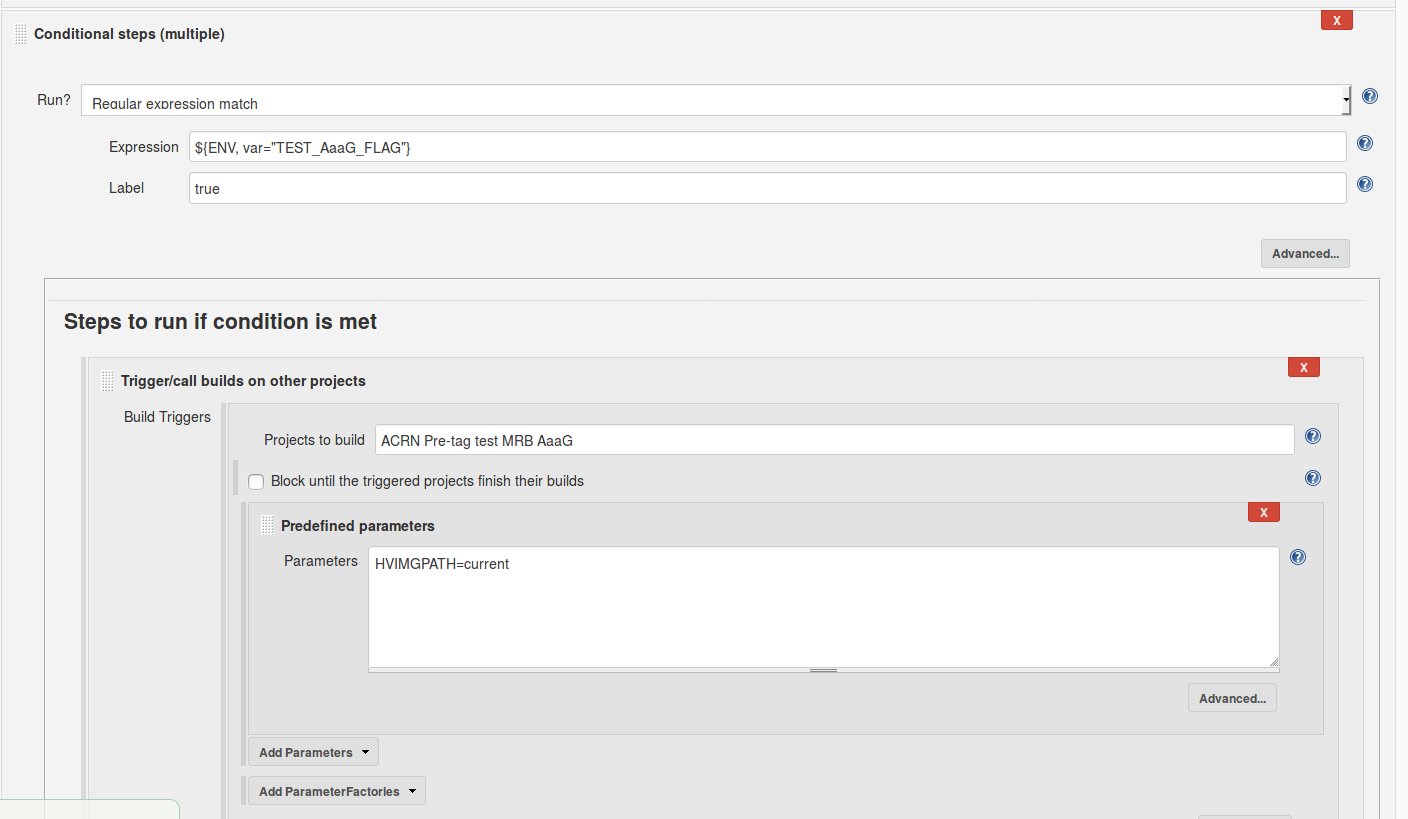
B\_VERSION=B2

这样我们在该Job中就有了一个内置变量 B\_VERSION

这里还介绍一种根据参数来控制是否trigger子job:

我们使用Conditional BuildStep Plugin来实现这一功能. 这是build option中有sub-option

如下图所示:



Conditional steps (multiple)

Run? : 执行一下任务的条件, 这里选择 Regular expression match

Expression和Label, Expression中的正则表达式与Label中的值相等这执行下面的任务, 否则不执行.

Steps to run if condition is met; 如果前面的条件满足, 则执行.

这里是trigger一个其他的project:

Projects to build: 要trigger的sub-project 的name

Predefined parameters: 要传递给sub-project的参数.

已经创建的Jenkins project

acrn-hypervisor-postmerge-build-image: 该JOB 是定时JOB. 即通过Poll SCM 定义每一分钟去GitHub检查acrn-hypervisor 是否有更新. 如果有则trigger一个build. 该JOB运行在10.239.147.197上. 主要的script 在/root/workspace/script/ acrn-hypervisor-postmerge-build-image/ 中. 该JOB有3个用于测试的sub job. 这3个job是并行执行的.

acrn-hypervisor-premerge-build-image: JOB是一个定时JOB, 没隔一分钟会通过GitHub Pull Request Builder 插件检查是否有新的PR提交. 然后根据新的PR的提交trigger 一个build. 会下载PR到最新的acrn-hypervisor sources code上.JOB运行在10.239.147.197 和10.239.147.197上.JOB有3个用于测试的sub job. 这3个job是并行执行的.

[acrn-kernel-postmerge-build-image](http://acrn-jenkins.sh.intel.com:8080/view/CI%20View/job/acrn-kernel-postmerge-build-image/): 同acrn-hypervisor-postmerge-build-image, 只不过该JOB是管理acrn-kernel 的postmerge. 运行在10.239.147.186上.

acrn-kernel-premerge-build-image: 同acrn-hypervisor-premerge-build-image. 只不过该JOB是管理acrn-kernel的premerge. 运行在10.239.147.197 和10.239.147.197上.

需要注意的是前面两个Job中: 我们在Jenkins job的配置中使用Git 插件下载acrn-hypervisor source code. 在script 的脚本中去下载或者更新acrn-kernel source code.后面的两个Job.我们使用Git 或者GitHub Pull Request Builder 下载管理acrn-kernel.

acrn-daily-release-build: B\_VERSION 参数, B1 版本还是B2版本; RELEASE\_OPION参数,是编译release还是debug版本; E2E\_ACRN\_BUILD\_ID 参数, 编译要使用的sos\_root.img和partition\_desc.bin的版本; TEST\_AaaG\_FLAG和TEST\_LaaG\_FLAG则是是否测试MRB的aaag image和laag image

acrn-mrb-engineer-build: 工作在GitHub的acrn-hypervisor 和acrn-kernel 上. 这里的参数有: HV\_COMMIT\_ID: 我们需要回退的acrn-hypervisor 的版本. 默认值是latest, 即最新的source code; HV\_PATCH\_LINKS: 需要debug的hypervisor 的patch 的链接, 可以是多个patch, patch link之间用逗号隔开. 当是多个patch时,按照填写的顺序apply. KERNEL\_COMMIT\_ID:作用与HV\_COMMIT\_ID类是,用来回退acrn-kernel的版本. KERNEL\_PATCH\_LINKS功能则是下载acrn-kernel的patch. 需要注意的是这两个LINKS的参数可以是PR的链接,也可以是developer自己的仓库的patch的链接;   
CLEARLINUX\_RELEASE\_ID:编译时需要用到的sos\_root.img 和partition\_desc.bin 的版本. 我们从<https://ubit-artifactory-sh.intel.com/artifactory/list/clearlinux-sh-local/GP2.0/> 处下载.这个链接是需要权限的, 目前使用的是wenlin的权限. 账号和密码保存在/root/workspace/clearlinux-release-images/account\_information.conf, 下载的IMG也保存在了这个目录下, 默认值是latest, 即下载最新的 IMG; IMG\_RELEASE\_VERSION: 编译的版本是DEBUG version还是RELEASE version.

MRB的其他几个engineer build, 用的sos\_root.img和partition\_desc.bin也是存在/root/workspace/clearlinux-release-images/ 中.