基于 Docker、Kubernetes 实现高效可靠的规模化 CI/CD 流水线的搭建

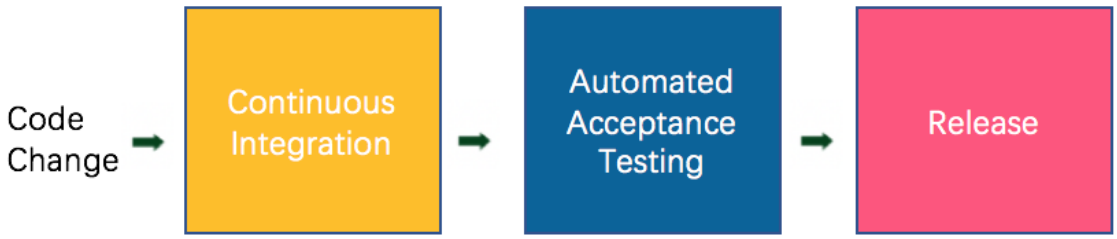
高效可靠的CI/CD流水线是一个IT组织实现软件服务快速交付的基础，现如今大量企业采用jenkins集群来搭建其交付流水线。然而，如何管理大量Jenkins Slave的差异化？如何简单快速实现Jenkins能力的横向扩展？如何实现流水线的高可用？如何有效利用闲置的Jenkins Slave资源？上述这些问题一直困拢着集群管理员，近两年随着虚拟化技术突飞猛进的发展，Docker, Kubernetes 等现代化工具彻底颠覆了交付团队的交付流程，同时也为CI/CD流程水线的搭建与管理提供了全新思路。

Setup

目前流行的CI工具很多，鉴于本文讨论CI/CD流水线在企业中的应用，考虑到企业不会有意愿将源代码的访问权开放给第三方，像Travis CI等这些基于SaaS的CI工具自然被pass，由于Jenkins在CI领域的主导性地位，所以本文的CI工具只涉及Jenkins。另外，本文默认您对Jenkins, Docker, Kubernetes 等工具有一些基础的了解。

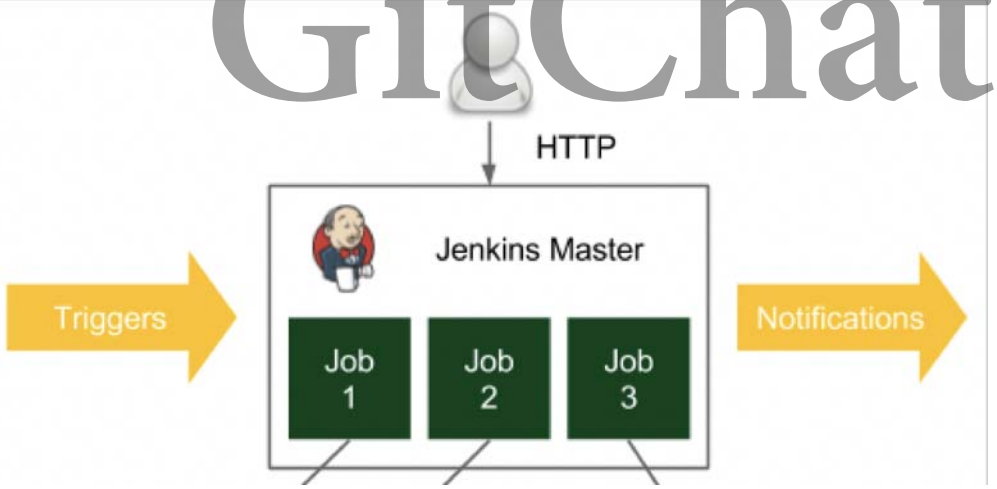
持续交付流水线简介

简单介绍一下持续交付流水线，如下图所示，简单来说流水线工作流程是这样的，当代码库有代码变更的时候持续集成服务器（ Jenkins ）会监听到代码变更并自动触发第一个阶段 — 持续集成阶段，这个阶段做的事情是自动构建，单元测试，静态代码分析以及生成报告。如果所有步骤都如预期成功通过的话会自动进入下一个阶段 — 自动化测试阶段，在跑自动化测试套件之前，首先要把成功通过第一阶段的产出物（ artifact), 如果选



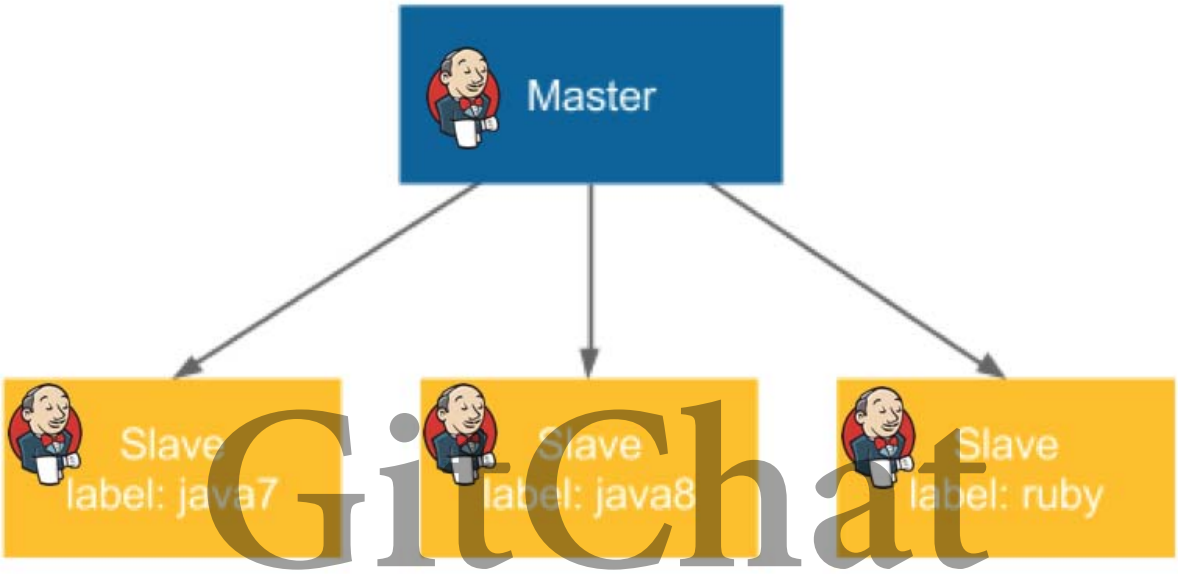
实践与痛点

上述这个过程实现起来需要这样做，首先需要把上图中各个阶段的工作脚本化，说具体一点就是需要写一个构建脚本来完成编译，单元测试，静态代码分析，生成报告等步骤，从而完成持续集成阶段的工作，然后是自动化测试脚本，这个脚本可以触发自动化测试套件并生成相关报告，最后需要写一个部署脚本，用于将持续集成阶段的产出物部署到测试环境上（当然最后的发布阶段也会重用这个脚本），这里需要注意的是要确保部署都是可重复的，重复部署同一个产出物N次的效果与只部署一次的效果相同，也就是大家常说的幂等。接下来就轮到Jenkins出场了，首先我们需要配置一下Jenkins来监听代码库的变更，这就意味着只要有代码迁入就会触发相对应的流水线，然后我们用Jenkins job或pipeline将这几个阶段的脚本串联起来（下图是以job为例），这样一个简单的CI/CD流水线就搭建完成了。



阶段的脚本将源代码从代码库中迁出，编译，单元测试，静态代码分析以及生成报告，Job2首先将持续集成阶段的产出物部署到测试环境并运行自动化测试套件，生成报告并标记产出物，Job3用于按需发布。每个Job所运行的脚本依赖的语言或运行环境会有所不同，可以通过label方式选择到相应的Slave上运行。

但在企业级大规模的应用CI/CD流水线时，由于企业中多团队多产品的存在，不同的团队会根据产品自身的特点来选择不同的技术实现方式，这就意味着产品实现语言会有很多种，比如java, C#, ruby, python, nodejs …那么相对应的CI/CD流水线就需要提供所有语言的编译环境并安装相关的依赖包，当然为了减少依赖，避免冲突以及更好的管理这些编译环境聪明的管理员会选择让每一个slave只能运行特定编程语言的Job, 如下图所示：



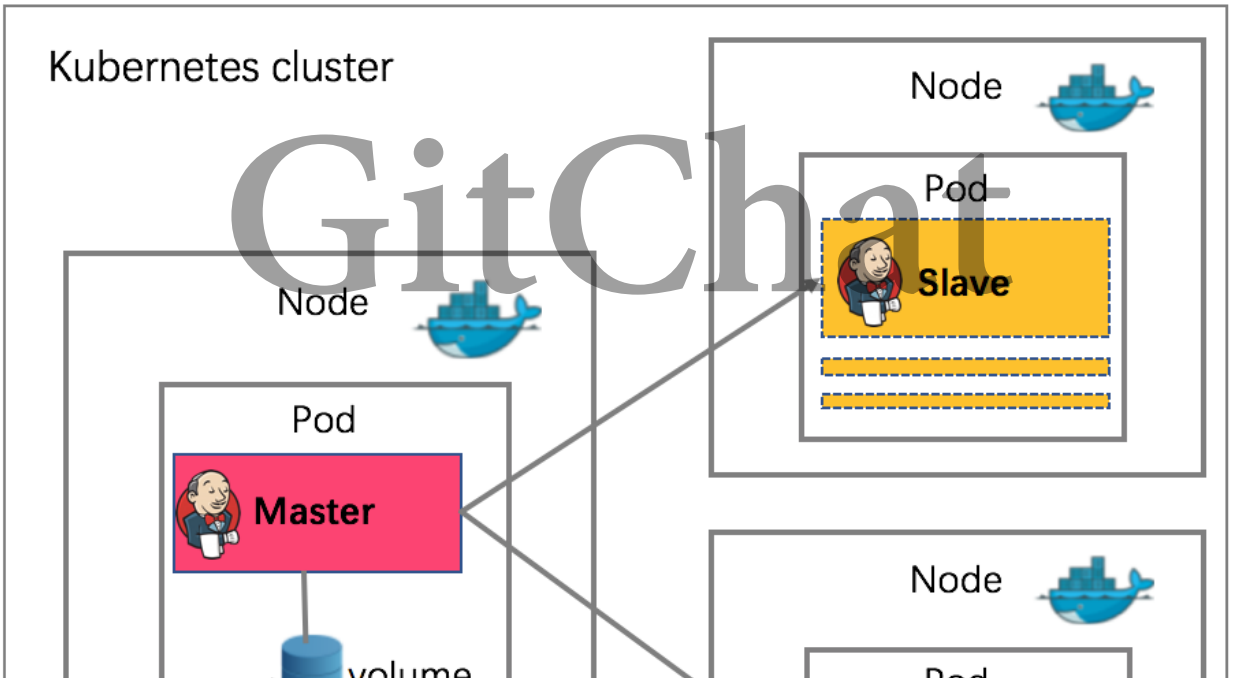
上图这个Jenkins集群Maser只用来调度和收集log，所有的Job都会由Master根据不同的label导流到对应的Slave上运行。这种集群的实现方式是我们在VM时代不得已的选择，虽然能解决大部分问题，但也带来了很多困扰。

- 单点依赖
  Jenkins Master成为单点，一旦Jenkins Master down机，那将是灾难性的，整个CI/CD流水线都将处于不可用的状态。

- 资源浪费
  每一台Jenkins Slave Server都是一台实实在在运行的VM, 当Slave Sever空闲时也不能将它所占用的资源释放，因为随时可能需要这个Slave完成相关的Job。

解决痛点

下面我们来看一下虚拟化技术带来了的福音，下图是一个基于Kubernetes, Docker搭建起来的Jenkins集群，为了避免混淆我略去了Kubernetes集群中的Master node。我们看到Jenkins Mater以Docker container的形式运行在Kubernetes一个Node上并将所有Jenkins相关数据存储到一个volume中，Jenkins Slave也以Docker container的形式运行在各个Node中，之所以用虚线来表现Slave是因为Slave不是一直存在的，它会被动态的按需创建并自动删除。简单介绍一下这种动态创建注册Slave的方式，它的工作流程是，当Jenkins Master收到一个build的请求时，会用按照label的要求动态的创建一个运行在Docker container中的Jenkins Slave并注册到Master上, 然后运行相应的Job，当Job运行完成后这个Slave会被注销，所在的Docker container也会被自动删除。



volume attach给新创建的Docker container，从而保证不会丢失任何数据，实现了Jenkins集群的高可用性。

- 自动伸缩
  由于每一次运行Job时，Jenkins Master都会动态创建一个Jenkins slave，Job完成之后Slave会被注销所在的Docker container也会被自动删除，所占用的资源就会被自动释放。也就是说当同时请求的Job数量越多，生成的Slave container就会越多，占用的资源也就越多，反之亦然，而且这种动态伸缩是完全不需要人为干预的。

- 完全隔离
  由于每一次运行Job都是在一个全新的Jenkins slave中运行，避免了同时运行的Job与Job之间发生冲突的可能性。

- 容易维护
  对比之前每一个Jenkins Slave是一台固定VM的做法，以这种方式搭建的集群维护的不再是固定的VM而是创建动态Slave所需要的Docker image，我们可以很容易通过Docker File来build适用于我们自己的Docker image并将它们存储在私有的Docker registry中，非常易于维护。

- 容易扩展
  当我们发现Jenkins的Queue中存在大量等待执行的Job是因为kubernetes集群的资源不足时，能够很容易的初始化一个kubernetes node并将它添加到集群中来，实现横向扩展非常的方便。

简单实现

下面我们来完成一个简单的实现：

1. 首先你需要安装一个Kubernetes cluster，请参考https://kubernetes.io/docs/setup/
   Kubernetes安装完成之后，首先需要用下面两条命令和文件部署一个Jenkins到Kubernetes集群：

```
[redacted]@jasonk8smaster:~$ cat jenkins-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: jenkins
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
      - name: jenkins
        image: jenkins
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
          name: web
        - containerPort: 50000
          name: agent
        volumeMounts:
          - name: jenkins-home
            mountPath: /var/jenkins_home
      volumes:
        - name: jenkins-home
          emptyDir: {}
```

这很重要

```
[redacted]@jasonk8smaster:~$ cat jenkins-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
      app: jenkins
  name: jenkins
spec:
  type: NodePort
  ports:
  - port: 8080
    targetPort: 8080
    name: web
  - port: 50000
    targetPort: 50000
    name: agent
  selector:
    app: jenkins
```

tip: 标红的部分很重要，开放8080端口是用来访问Jenkins web portal用的；而动态创建的Jenkins slave会默认通过50000(可修改）端口与master建立连接。

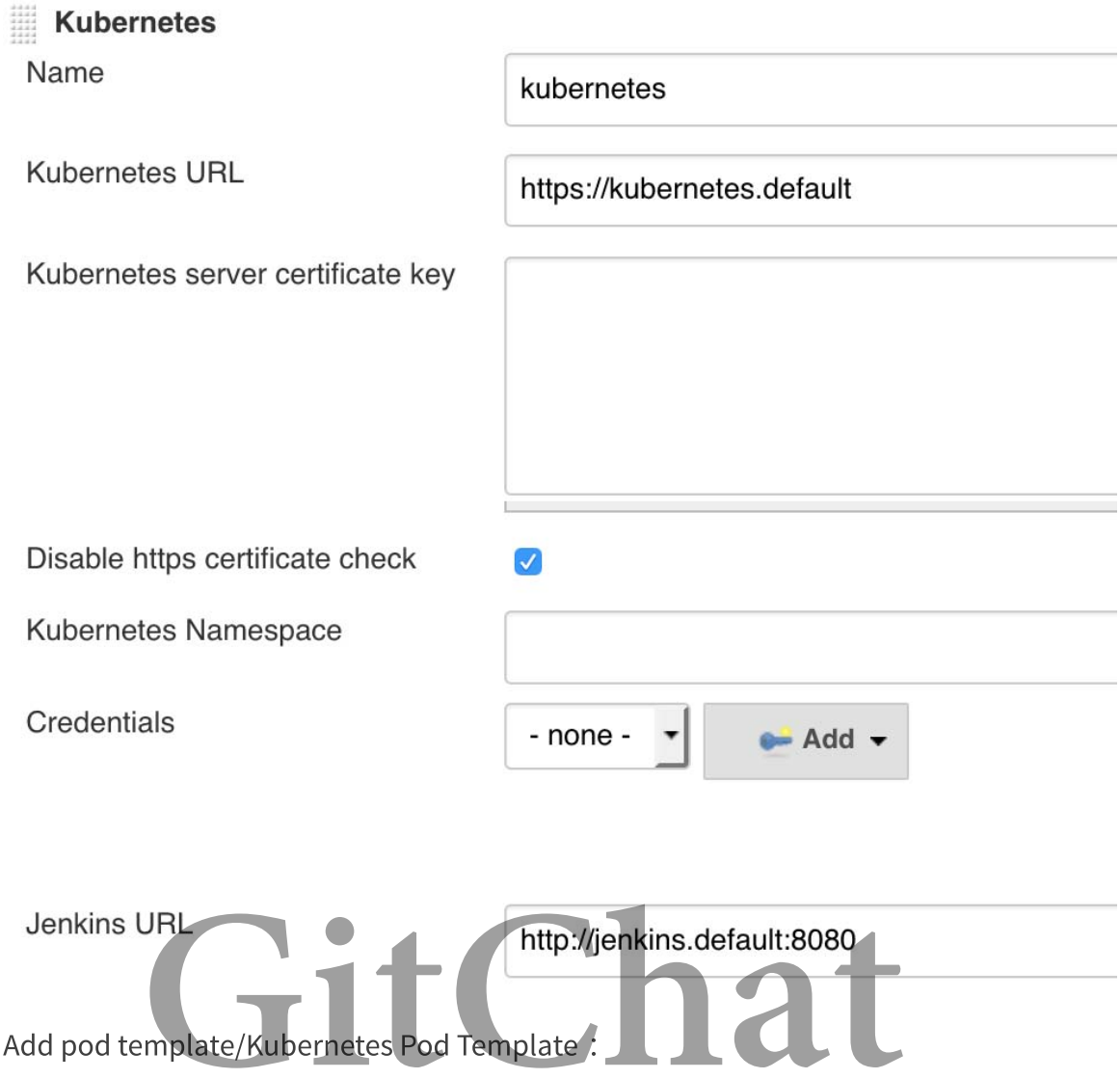
检查jenkins安装运行情况：

```
[redacted]@jasonk8smaster:~$ kubectl get po
NAME                              READY     STATUS    RESTARTS   AGE
jenkins-2191298921-9fz37          1/1       Running   0          25s
```

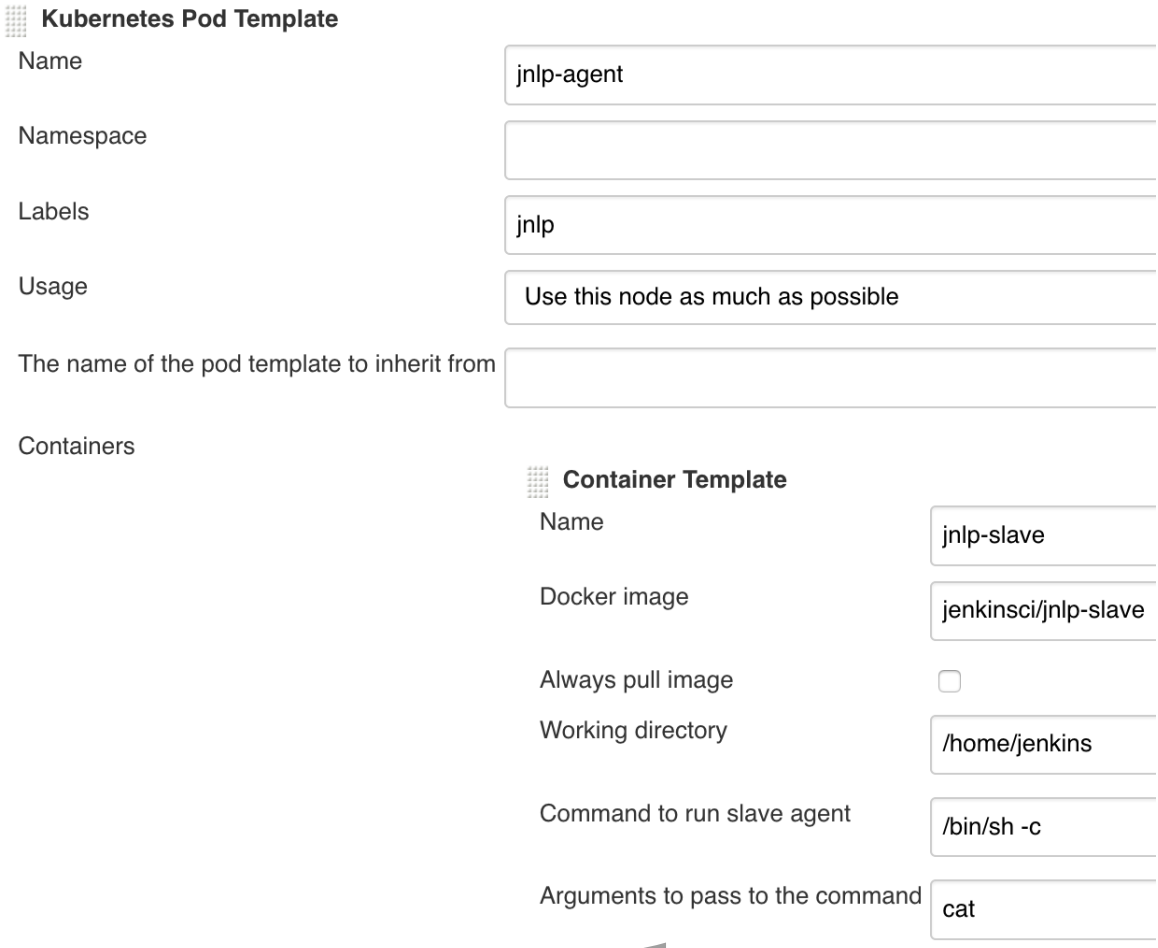
2. 查看Jenkins log，用管理员密码登录Jenkins并安装Kubernetes plugin。

3. 配置Kubernetes cloud

   Manage Jenkins / Configure System/ Add a new cloud/ Kubernetes：



Add pod template/Kubernetes Pod Template：



点击Save之后大功告成。

牛刀小试

下面我们来测试一下这个动态注册Slave的Jenkins集群是否工作正常，首先登录Jenkins创建一个简单的free style job，指定这个Job只能在Label为“jnlp”的agent上运行。点击build now，你会发现奇迹发生了，原来没有注册任何Slave的Jenkins动态的创建一个Slave并注册到Master上，然后运行相应的Job，当Job运行结束后这个Slave被自动清除

PS：这只是一个简单的实现，在企业的实践中，我们需要不同的build环境，需要我们基于jenkinsci/jnlp-slave这个Image构建我们自己的Jenkins slave Image并保存到私有的Registry中，相对应的Kubernetes需要从私有Registry拉取Image。

参考文献：

- https://kubernetes.io/
- https://www.docker.com/
- https://github.com/jenkinsci/kubernetes-plugin
- https://kumorilabs.com/blog/k8s-6-integrating-jenkins-kubernetes/