




基于Kubernetes的Jenkins构建集群实践

 李华强 乐视致新

 2017.09.17



目 录

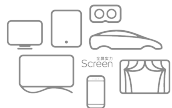


1 Jenkins分布式构建架构

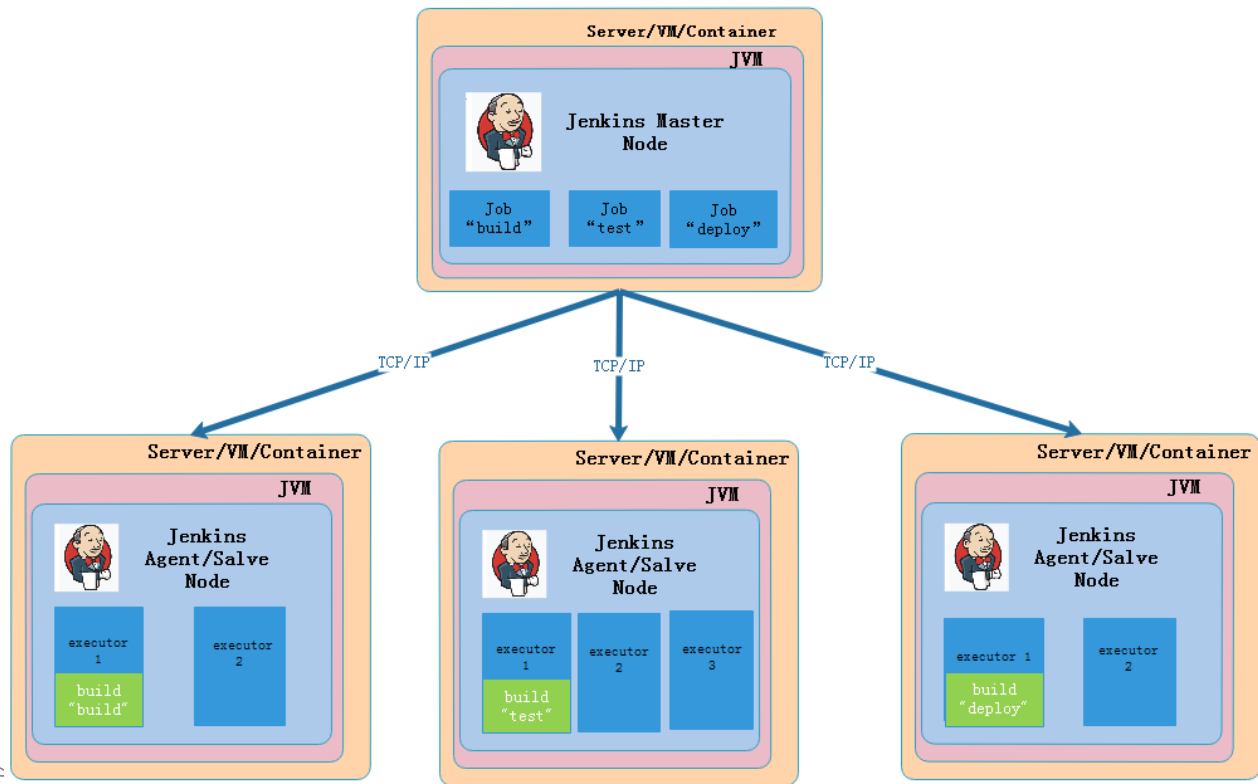
2 基于Lable的Slave集群管理

3 基于Docker插件的容器化实践

4 基于Kubernetes的容器化实践

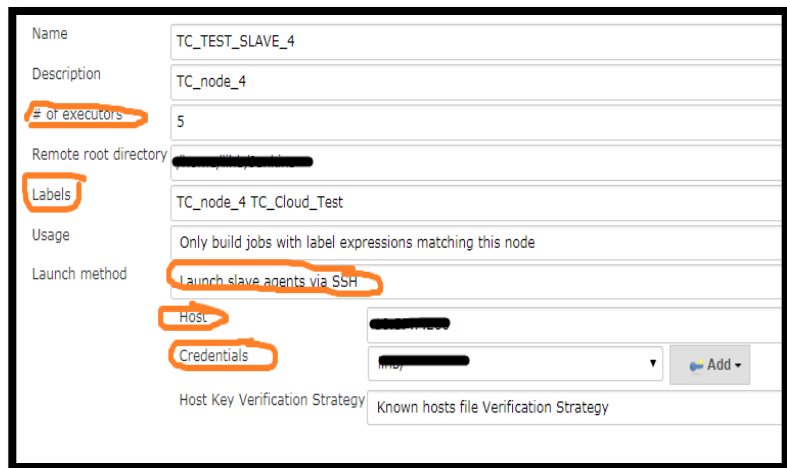


Jenkins分布式构建架构



Jenkins Slave 连接方式

➤ Launch slave agents via **SSH**

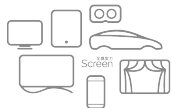


The screenshot shows the Jenkins 'New Slave' configuration page. The 'Launch method' is set to 'Launch slave agents via SSH'. Other fields include 'Name' (TC_TEST_SLAVE_4), 'Description' (TC_node_4), '# of executors' (5), 'Remote root directory' (redacted), 'Labels' (TC_node_4 TC_Cloud_Test), 'Usage' (Only build jobs with label expressions matching this node), 'Host' (redacted), 'Credentials' (redacted), and 'Host Key Verification Strategy' (Known hosts file Verification Strategy). Several fields are circled in orange: '# of executors', 'Labels', 'Launch method', 'Host', and 'Credentials'.

➤ Launch agent via **Java Web Start (JNLP)**



The screenshot shows the Jenkins 'Agent test-jnlp' page. It displays an error message: 'Ping response time is too long or timed out.' Below this, it provides instructions on how to connect the agent to Jenkins. The instructions include a 'Launch' button, a command line for running the agent from the command line, and a command line for running the agent headlessly. The 'Projects tied to test-jnlp' section shows 'None'.



/ Jenkins 调度策略

➤ 用户视角

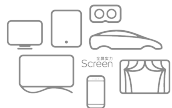
- ❑ 直接使用slave name，构建使用指定slave
- ❑ 使用label，构建多数使用同一个的slave，偶尔使用其他slave

➤ 系统实现

- ❑ consistent hashing algorithm (node , available executors , job name)
- ❑ 每一个Job都有一个对应所有Node的优先级列表

➤ 其他插件

- ❑ Least Load (<https://plugins.jenkins.io/leastload>)
- ❑ Commercial plugin [Even Scheduler Plugin](#)



目 录

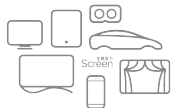
1 Jenkins分布式构建架构



2 基于Lable的Slave集群管理

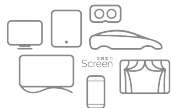
3 基于Docker插件的容器化实践

4 基于Kubernetes的容器化实践



/ Android产品复杂环境

业务产品差异性	Mobile, TV, Box, IOV, VR 等
多种构建类型	Daily build , Test build , Continuous Build , APK 编译等
代码量大	50GB +
编译空间大	100-200GB
编译时间长	2-3小时 (24线程)
业务间相对独立	专属编译服务器 , 独立的构建环境
服务器差异	版本 , CPU , Memory , Disk等



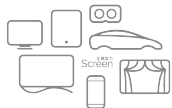
/ Jenkins使用中暴露的问题

表象问题：

- Slave很多，空闲的Slave也很多
- 队列中等待构建任务很多
- 一些构建失败，暴露workspace空间不足问题

原因分析：

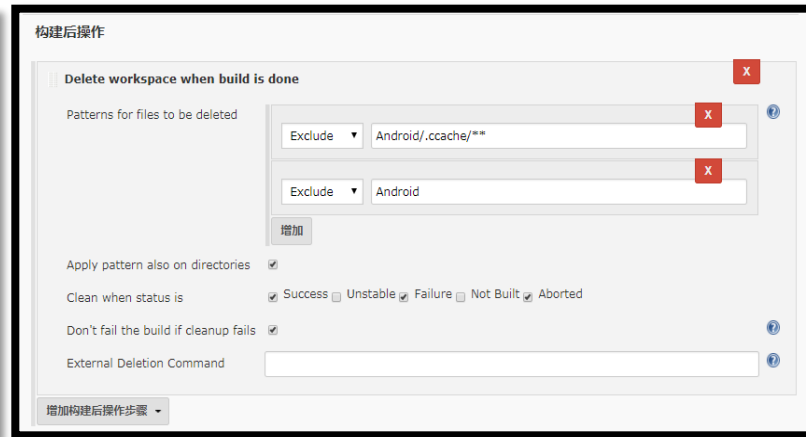
- 业务间，编译环境不统一，不能跨业务共享
- 业务内，特定jobs直接绑定特定slave，并发量受限于executor数目
- slave上构建workspace的遗留，占用大量空间
- 业务量增大，新建Jobs增多，瓶颈出现



优化改进

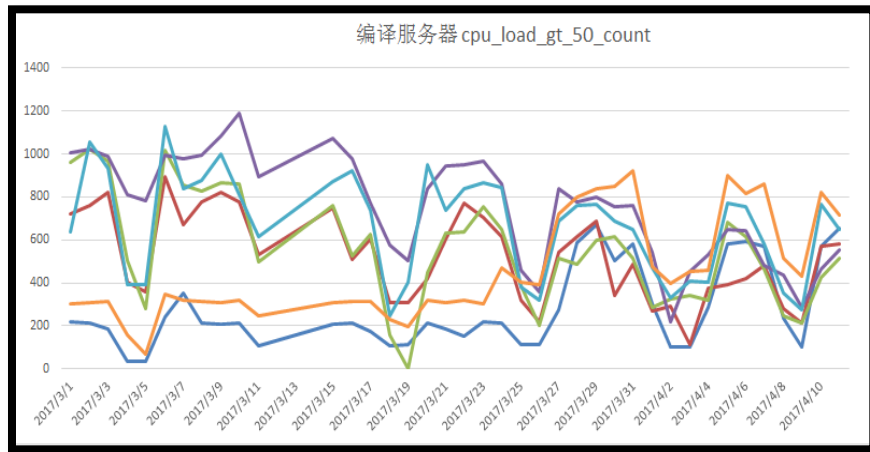
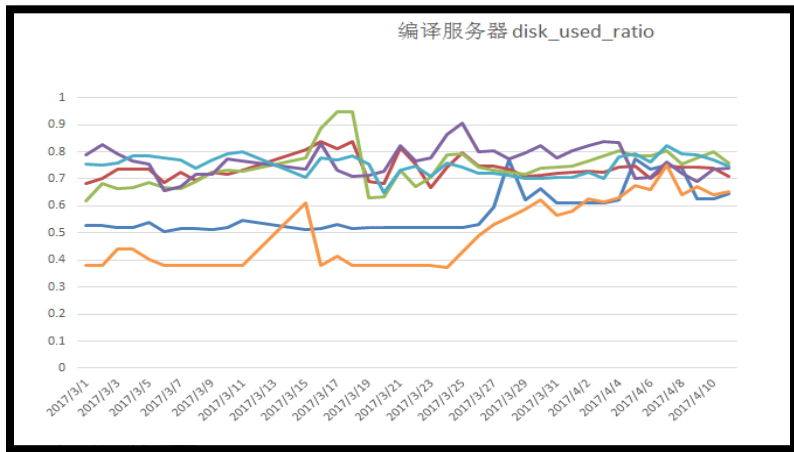
优化：

- 同质slaves添加相同Label，同类型jobs使用Label进行构建
- 适量增大slave的executor数目
- 定时扫描清理slaves上的废弃jobs的遗留workspace
- 业务jobs配置workspace清理规则



优化效果

- 并发能力增强，队列中构建任务缓解
- Slave空间有效利率提高，空间不足造成的构建失败大大减少
- Slave集群综合资源利用率提高（Disk，CPU），趋于一致性



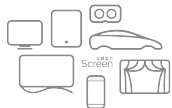
/ Lable方案反思

益处：

- Slave资源池化，整体资源利用率提高
- 并发量增大（受所有slave的executor数量限制）
- Slave的管理对Job配置透明化（Job配置Label使用）

局限性：

- Jenkins内置的调度策略，资源利用不均衡
 - ❑ 同一个job的多次构建倾向于使用同一个slave
 - ❑ 同一个job的并发构建，往往使用同一个slave，资源竞争造成构建时间增加等
 - ❑ 某几个jobs构建，往往使用同一个slave，资源竞争造成构建时间增加等
- 环境隔离问题
 - ❑ 不同类型的job资源需求不同，环境复用后资源调度是问题，加大管理成本
 - ❑ 环境不统一，业务调整，资源再分配要回炉重构（安装对应的工具系统等）
- 基础设施一致性问题
 - ❑ 多个任务跑在同一台资源上的潜在风险和冲突，不可变基础设施
 - ❑ 环境问题导致的CI信任问题，代码没错，是环境问题



目录

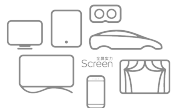
1 Jenkins分布式构建架构

2 基于Lable的Slave集群管理



3 基于Docker插件的容器化实践

4 基于Kubernetes的容器化实践

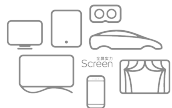


/ APK编译需求

编译现状：

- 单点编译服务器支持所有APK编译构建
- 服务器性能比较差
- 环境依赖复杂，工具维护成本高
- 构建任务并发比较困难

何去何从？尝试容器化！



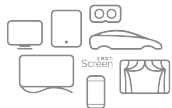
/ Docker image 固化编译环境

镜像包含哪些内容？

- 账号，权限
- JDK + SSHD （ Jenkins Slave 要求 ）
- 依赖工具（ 不同版本Android SDK/NDK, libs , tools等 ）

需要考虑的问题

- Docker images ，业务产品，不同版本工具集的关系
- Docker images 自动化构建、更新、上传、部署等



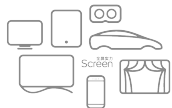
/ Jenkins 集成 Docker 插件

➤ 插件

- ❑ 使用版本 0.15.0 (SSH slave support)
- ❑ 最新 0.16.2 (JNLP slave support (Experimental))
- ❑ <https://wiki.jenkins.io/display/JENKINS/Docker+Plugin>

➤ 配置

- ❑ 1个或者多个Docker host 信息
- ❑ 每个Docker host 下关联1个或者多个Docker template (image) 信息



配置示例

Cloud

Docker

Name

Docker URL

Credentials

Connection Timeout

Read Timeout

Container Cap

Images

Docker Template

Docker Image

Instance Capacity

Remote Filing System Root

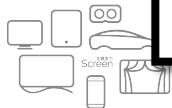
Labels

Usage

Launch method

Credentials

Host Key Verification Strategy



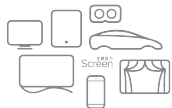
/ Docker插件反思

益处：

- 容器化（环境标准化，隔离性，版本，可移植性 等等）
- 容器Slave 按需弹性收缩，自动创建，使用，销毁
- 资源共享
 - ❑ 多个jobs使用不同容器slave可以运行在同个Docker host
 - ❑ 同一Job通过使用label，也可以让构建运行在不同Docker host上的容器slave上
- 并发量（插件设定cloud/template Capacity）

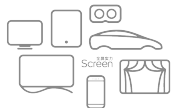
局限性：

- 插件配置中 Docker host 与Docker image的强耦合性，配置不方便
 - ❑ Docker host 数量很多的时候
 - ❑ Docker image 需要配置到多个Docker host上的时候
- 使用Jenkins内置的调度策略，资源利用不均衡
 - ❑ 相同Docker image 配置到多个Docker host上使用相同label的时候



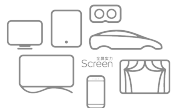
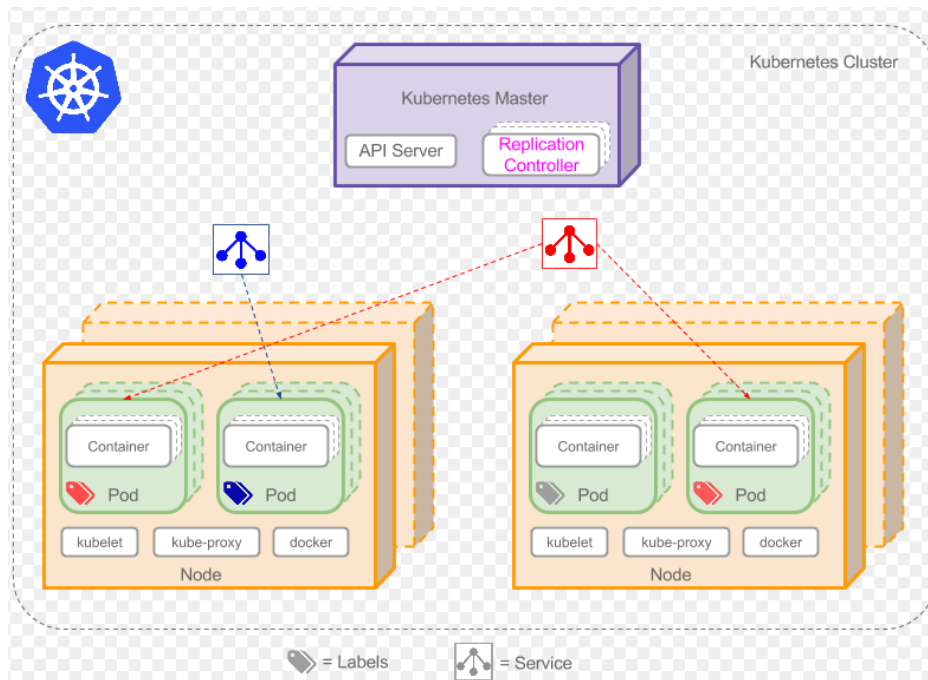
目 录

- 1 Jenkins分布式构建架构
- 2 基于Lable的Slave集群管理
- 3 基于Docker插件的容器化实践
-  4 基于Kubernetes的容器化实践



/ What is Kubernetes?

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
- <https://kubernetes.io/>

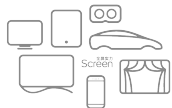


/ 业务需求

- 资源收缩严重，业务需要正常运转
- 容器化/标准化，构建环境隔离
- K8s 资源限制，调度策略，服务器资源共享

其他考虑因素

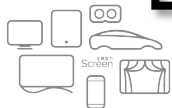
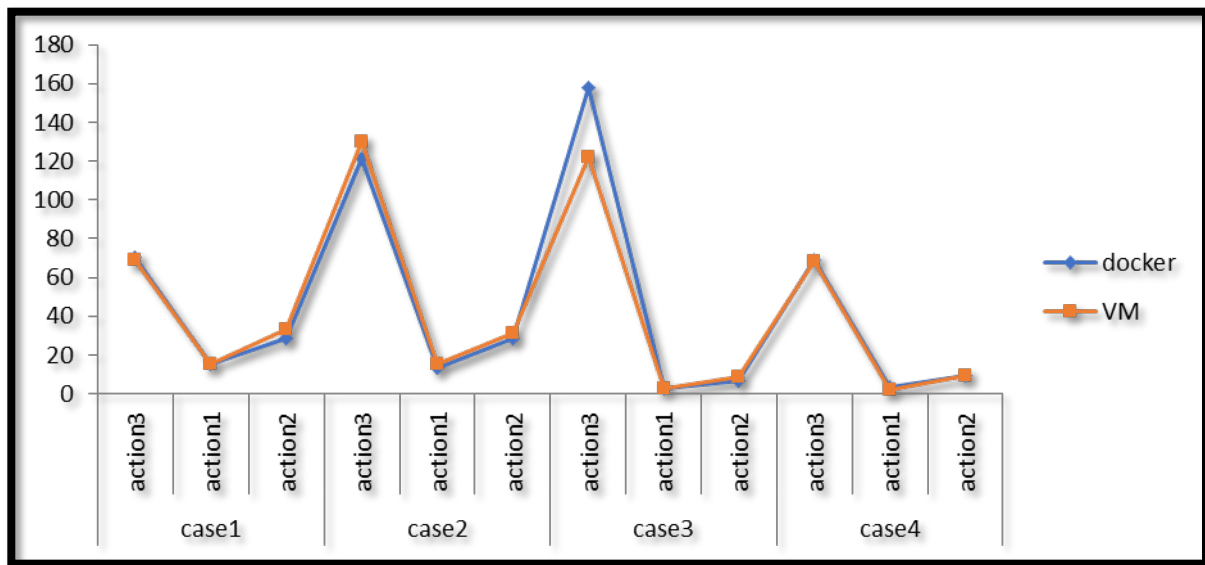
- 容器化构建性能对比
- 版本的选择
- Docker registry
- Docker image的构建更新自动化
- 构建环境工具依赖
- 构建优化（代码mirror，编译缓存等）



/ 数据验证可行性

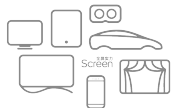
在不同场景条件下的性能对比测试得出：

使用容器编译性能基本一致



/ 选择版本

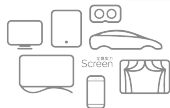
组件	版本	说明
Jenkins	2.46.2	Jenkins core
Kubernetes	1.5.4	主程序
Jenkins Kubernetes plugin	0.8	Jenkins 插件
Docker	17.03.1-ce	容器
Harbor	v1.1.2	私有镜像库
Flannel	0.5.5	网络组件
Etcd	3.1.6	数据库
Kubernetes-dashboard	v1.5.0	界面
kubedns	1.9	DNS组件
Heapter	v1.3.0	监控
influxdb	v1.1.1	监控数据库
grafana	v4.4.1	监控界面



/ Docker image的构建

- Dockerfile (Docker image 构建脚本)
- Ansible playbook (容器和K8s Node , 环境部署自动化)

```
1
2 ## Dockerfile example
3 FROM harbor.devops.leco.com:5000/test/tv:ansible
4 MAINTAINER Li huaqiang <lihuaqiang@le.com>
5 ADD ./docker /docker
6 WORKDIR /docker
7 RUN export TMPDIR=/var/tmp && ansible-playbook -v -i playbooks/inventories/docker playbooks/docker_container.yml
8
9 ##playbook example
10
11 ---
12
13 - hosts: localhost
14   roles:
15     - accounts
16     - base-dirs
17     - base-apt
18     - base-pkgs
19     - base-tools
20     - git-customized
21     - jdk-customized
22
```



/ 构建环境依赖

外部工具

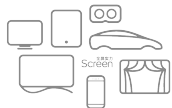
- 单一可信数据源，单独服务器管理，Ceph存储
- 所有k8s Node 通过NFS挂载
- 容器通过volume使用

代码mirror

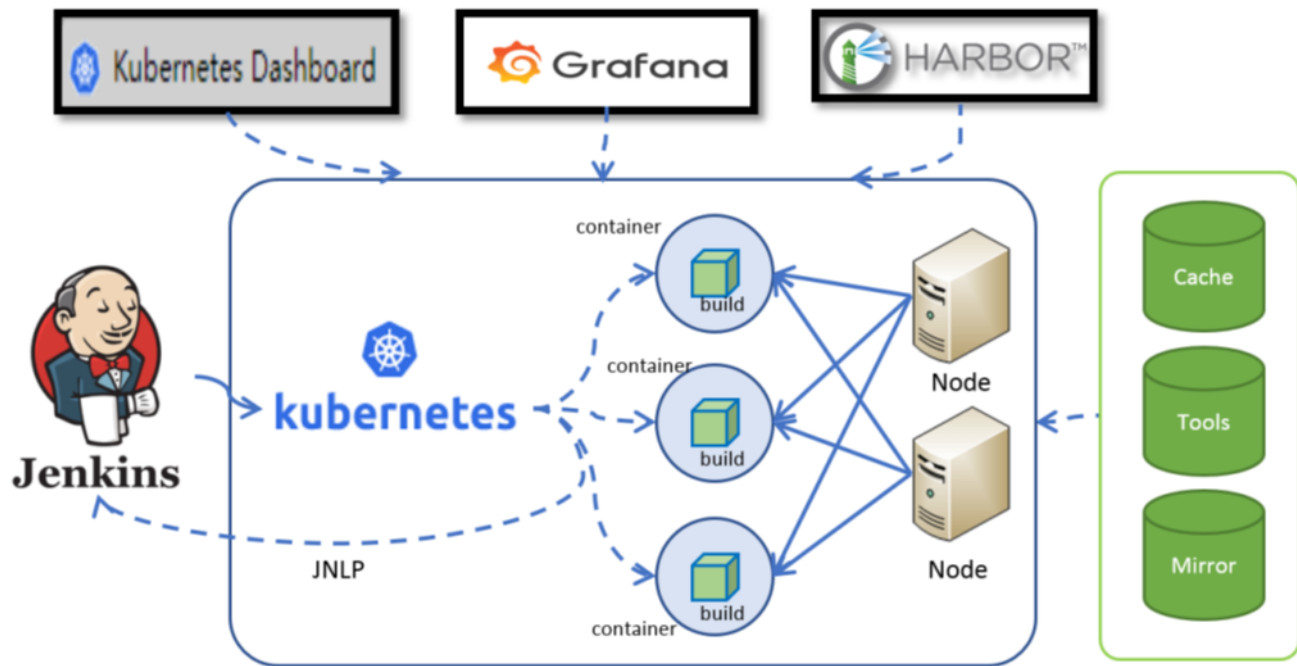
- 所有k8s Node 本地目录
- 容器通过volume使用
- Jenkinsjob自动化更新部署

编译cache

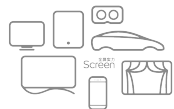
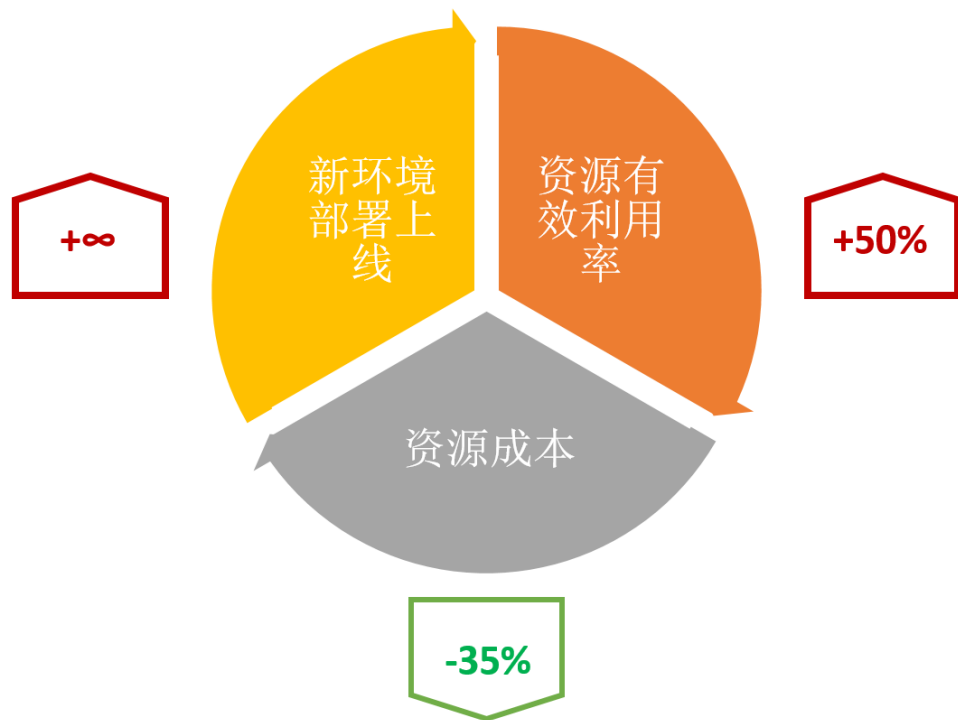
- 所有k8s Node 本地目录
- 容器通过volume使用
- 命中率统计和多基线复用



/ Jenkins集成k8s全景



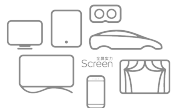
/ 实现效果



/ Kubernetes插件

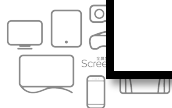
插件配置

- 配置 Kubernetes API URL
- 添加1个或多个kubernetes pod template (image)
- kubernetes pod template 的启动命令间接使用JNLP
- kubernetes pod template 可以配置资源限制



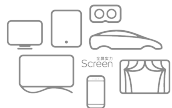
配置示例

Kubernetes	
Name	k8s production cluster
Kubernetes URL	http://10.100.92.76:8080
Kubernetes server certificate key	
Disable https certificate check	<input type="checkbox"/>
Kubernetes Namespace	default
Credentials	- none - Add
Jenkins URL	http://10.100.92.76:8080/
Jenkins tunnel	
Connection Timeout	5
Read Timeout	15
Container Cap	10
Images	
Kubernetes Pod Template	
Name	apk-slave-k8s
Labels	docker-apk-slave-k8s
Docker image	10.100.92.76:5000/citestr/apk-slave-k8s:3
Always pull image	<input type="checkbox"/>
Jenkins slave root directory	/root
Command to run slave agent	/usr/bin/jenkins-slave
Arguments to pass to the command	
Arguments to pass to the command. The slave secret and name are always added to the arguments.	
Max number of instances	5
Volumes	
Host Path Volume	
Host path	/data/cache/gradle
Mount path	/root/.gradle



/ 资源限制配置示例

Annotations	<div>Add Environment Variable -</div> <div>List of environment variables to set in slave pod</div> <div>Add Annotation -</div> <div>List of annotations to set in slave pod</div>
Run in privileged mode	
ImagePullSecrets	<div>Add Image Pull Secret -</div> <div>List of image pull secrets</div>
Service Account	
<u>Node Selector</u>	
<u>Request CPU</u>	5
<u>Request Memory</u>	25Gi
<u>Limit CPU</u>	12
<u>Limit Memory</u>	60Gi



/ 容器启动脚本示例

```
#!/usr/bin/env bash

# Usage jenkins-slave.sh [options] -url http://jenkins SECRET SLAVE_NAME
# Optional environment variables :
# * JENKINS_TUNNEL : HOST:PORT for a tunnel to route TCP traffic to jenkins host, when jenkins can't be directly accessed over network
# * JENKINS_URL : alternate jenkins URL

if [ $# -eq 1 ]; then

    # if `docker run` only has one arguments, we assume user is running alternate command like `bash` to inspect the image
    exec "$@"

else

    # if -tunnel is not provided try env vars
    if [ [ "$@" != *"-tunnel" * ] ]; then
        if [ ! -z "$JENKINS_TUNNEL" ]; then
            TUNNEL="-tunnel $JENKINS_TUNNEL"
        fi
    fi

    if [ ! -z "$JENKINS_URL" ]; then
        URL="-url $JENKINS_URL"
    fi

    exec java $JAVA_OPTS -cp /usr/share/jenkins/slave.jar hudson.remoting.jnlp.Main -headless $TUNNEL $URL "$@"
fi
```

这2个参数Kubernetes插件默认传递给启动参数

插件默认设置JENKINS_URL为POD环境变量

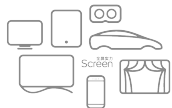
/ Kubernetes插件反思

益处：

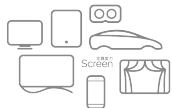
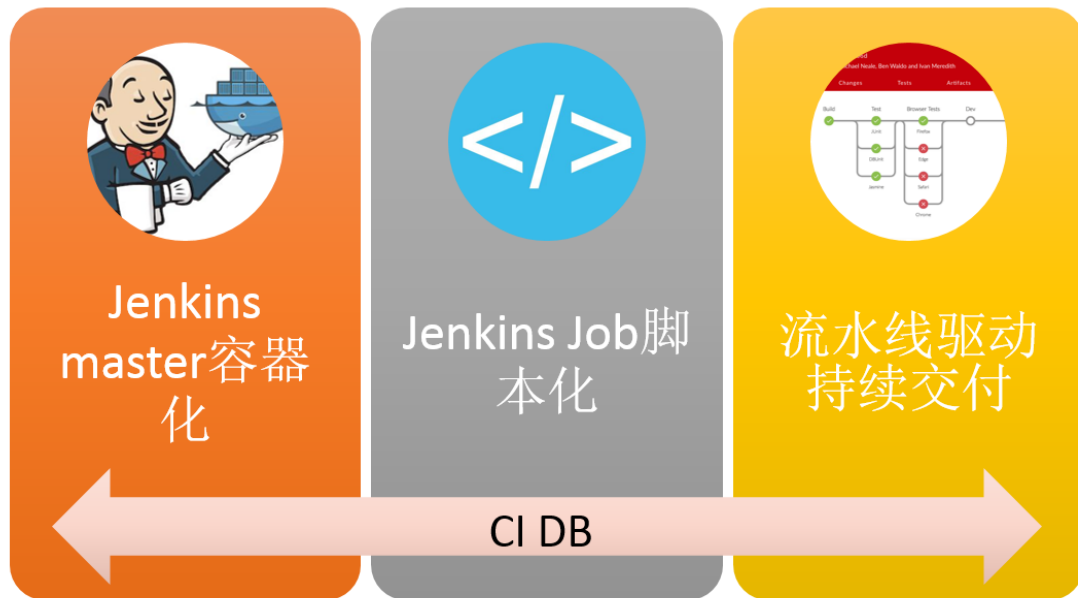
- 容器化（环境标准化，隔离性，版本，可移植性 等等）
- 容器Slave 按需弹性收缩，自动创建，使用，销毁
- 资源共享（所有的容器slave可以共享 Kubernetes cluster）
- 并发量（插件设定Cloud/Template Capacity）

局限性：

- Kubernetes Cluster 的高可用性
 - ❑ 单 master
 - ❑ 需要用户实现HA方案

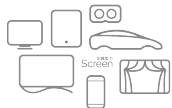


/ 不要停下持续改进的脚步



/ 简单总结

方案	效果	注意点
Lable	<ol style="list-style-type: none">1. Slave资源池化，整体资源利用率提高2. 构建并发量3. Slave的变动对Job配置透明化	<ol style="list-style-type: none">1. 环境一致性，相同label下slave要求同质2. Slave上jobs构建遗留workspace问题3. Jenkins默认调度策略的不完美性
Docker	<ol style="list-style-type: none">1. 容器化带来的好处（环境标准化，隔离性，版本，可移植性 等等）2. 容器Slave 按需弹性收缩，自动创建，使用，销毁3. 服务器资源共享4. 构建并发量设置	<ol style="list-style-type: none">1. Docker host 与Docker image的强耦合性，插件配置不方便性2. 使用Jenkins默认调度策略3. Docker registry、image管理等配套工作
Kubernetes	<ol style="list-style-type: none">1. Docker插件相同的优点2. 资源申请限制等配置，满足不同构建需求3. 使用Kubernetes 调度策略	<ol style="list-style-type: none">1. Kubernetes Cluster 的高可用性 问题2. 监控的重要性3. Kubernetes、Docker registry、image管理等配套工作



/ 问题？

