架构融合
云化共建

云原生DevOps在AI领域的实战

目录

- 什么是云原生
- 云原生业界通用DevOps玩法（行业对标）
- AI领域在DevOps错与对和软着陆的抓手
- 云原生容器化DevOps流水线在AI领域的实战

- 什么是云原生
- 云原生业界通用DevOps玩法（行业对标）
- AI领域在DevOps错与对和软着陆的抓手
- 云原生容器化DevOps流水线在AI领域的实战

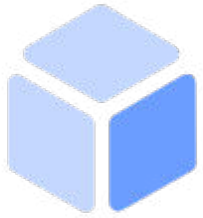
CNCF

- Cloud Native Computing Foundation
- 云原生计算基金会隶属于Linux基金会，于2015年成立
- 致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术。通过将最前沿的模式民主化，让这些创新为大众所用

CNCF对云原生的定义

- 云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中构建和运行可弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式API。
- 这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，元原生技术使工程师能够轻松地对系统做出频繁和可预测的重大变更。

云原生四大特征



微服务

容器化

DevOps

持续交付

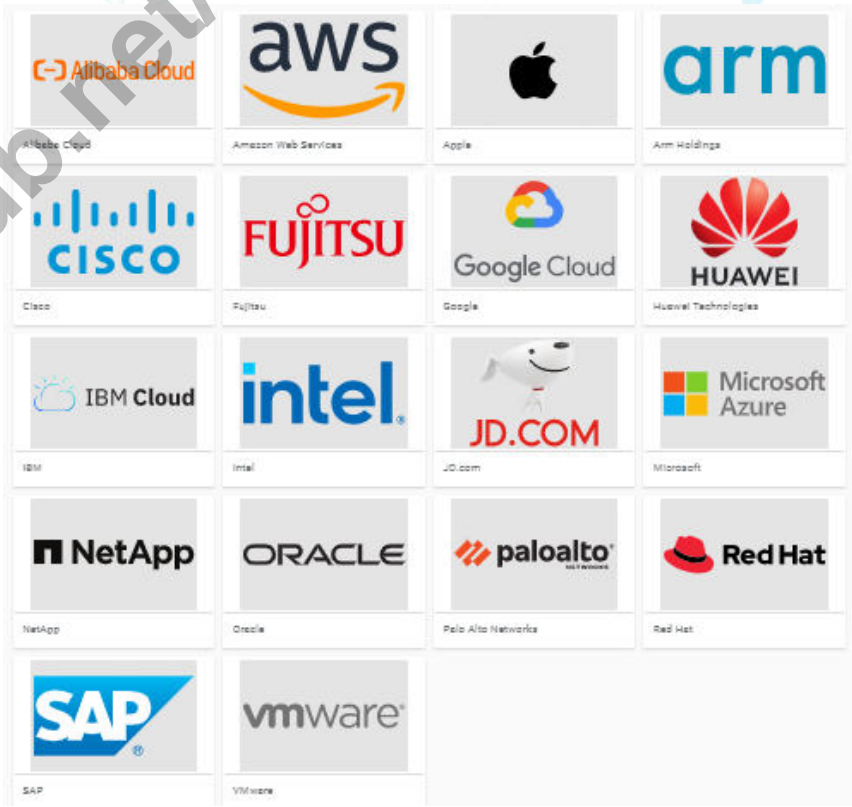
云原生社区发展现状

103,104

CNCF项目贡献者

157,193

会员数



- 什么是云原生
- 云原生业界通用DevOps玩法（行业对标）
- AI领域在DevOps错与对和软着陆的抓手
- 云原生容器化DevOps流水线在AI领域的实战

一千个读者就有一千个哈姆雷特

对标业界

业界各大标杆公司均已沉淀出自己的开发云平台，这也证明了开发云已经是业界通用玩法和最佳实践！

公司	产品	链接地址
AWS	开发人员工具	https://www.amazonaws.cn/products
阿里	云效	https://www.aliyun.com/product/yunxiao
腾讯	腾讯云开发者平台	https://dev.tencent.com
百度	效率云	http://xiaolvyun.baidu.com
华为	DevCloud	https://www.huaweicloud.com/devcloud
…	…	…

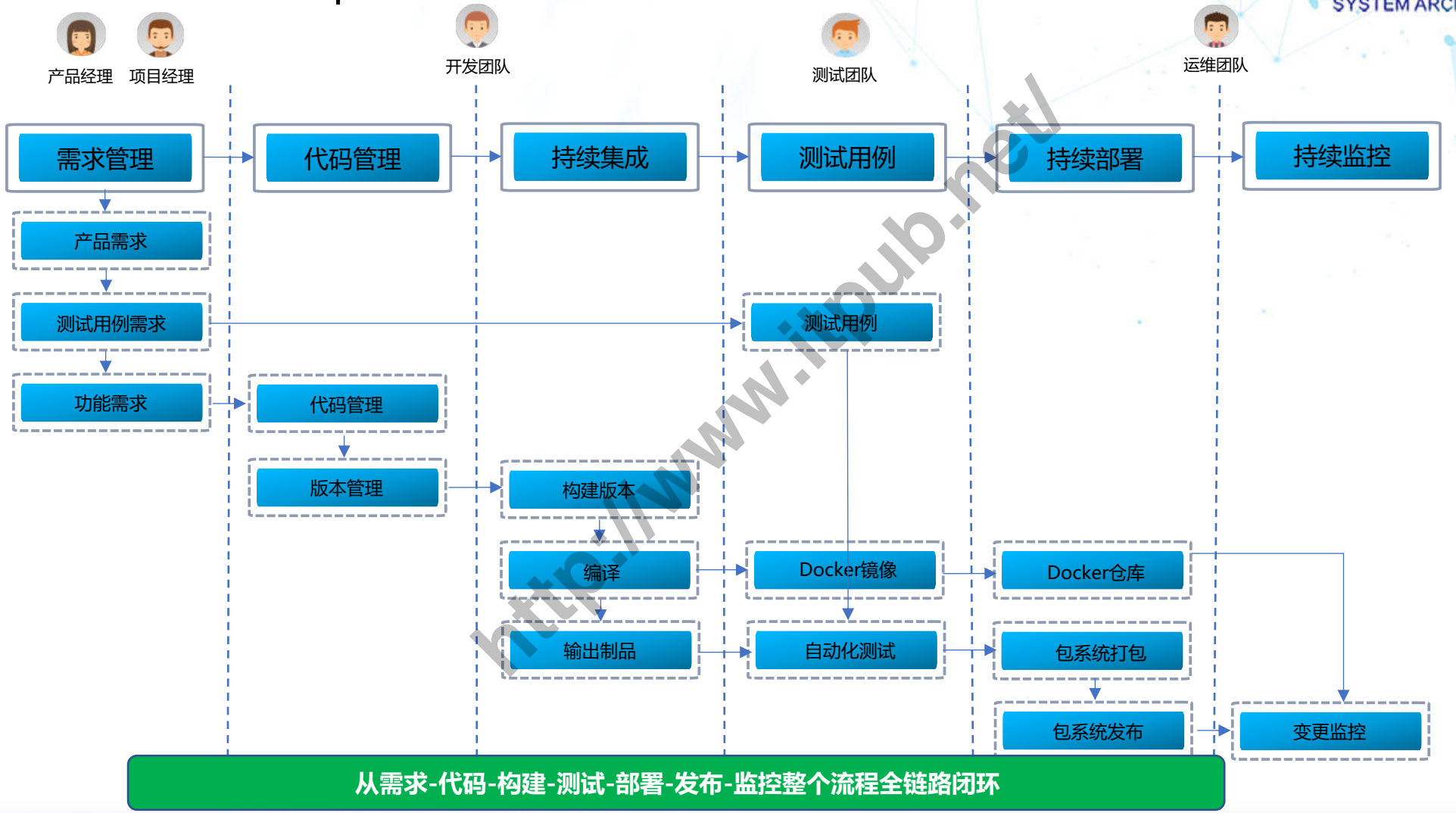
■ 从业界来看

从研发痛点出发，提供针对性解决方案，最大程度的实现降本增效！

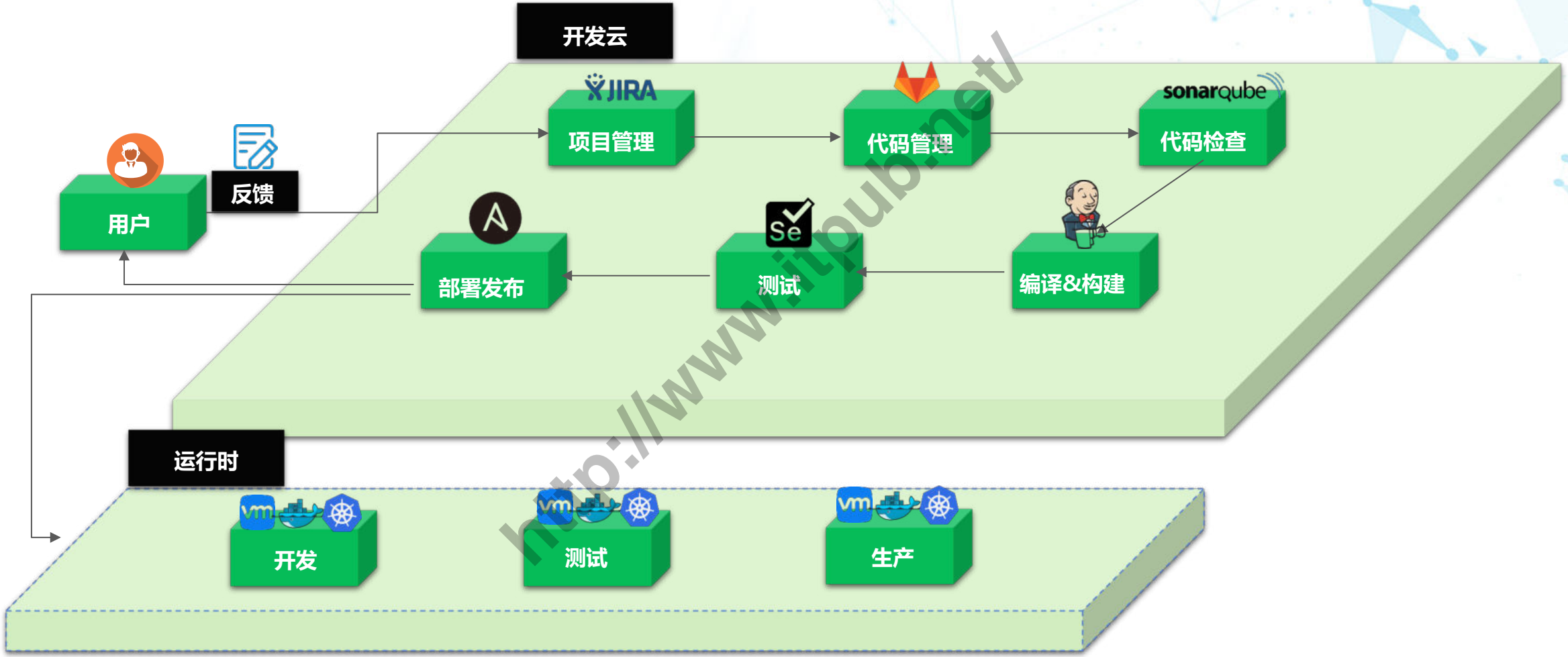
现状或痛点	解决思路
效率问题	资源入云，弹性伸缩
开发测试环境复杂	资源入云，环境标准化
基础设施维护问题	资源入云，高可靠的容灾备份
平台能力分散、冗余	融合与完善平台能力
产品线诉求	一站式云端DevOps平台
…	…

■ 从公司大背景和研发发展趋势来看

通用玩法 | 流程

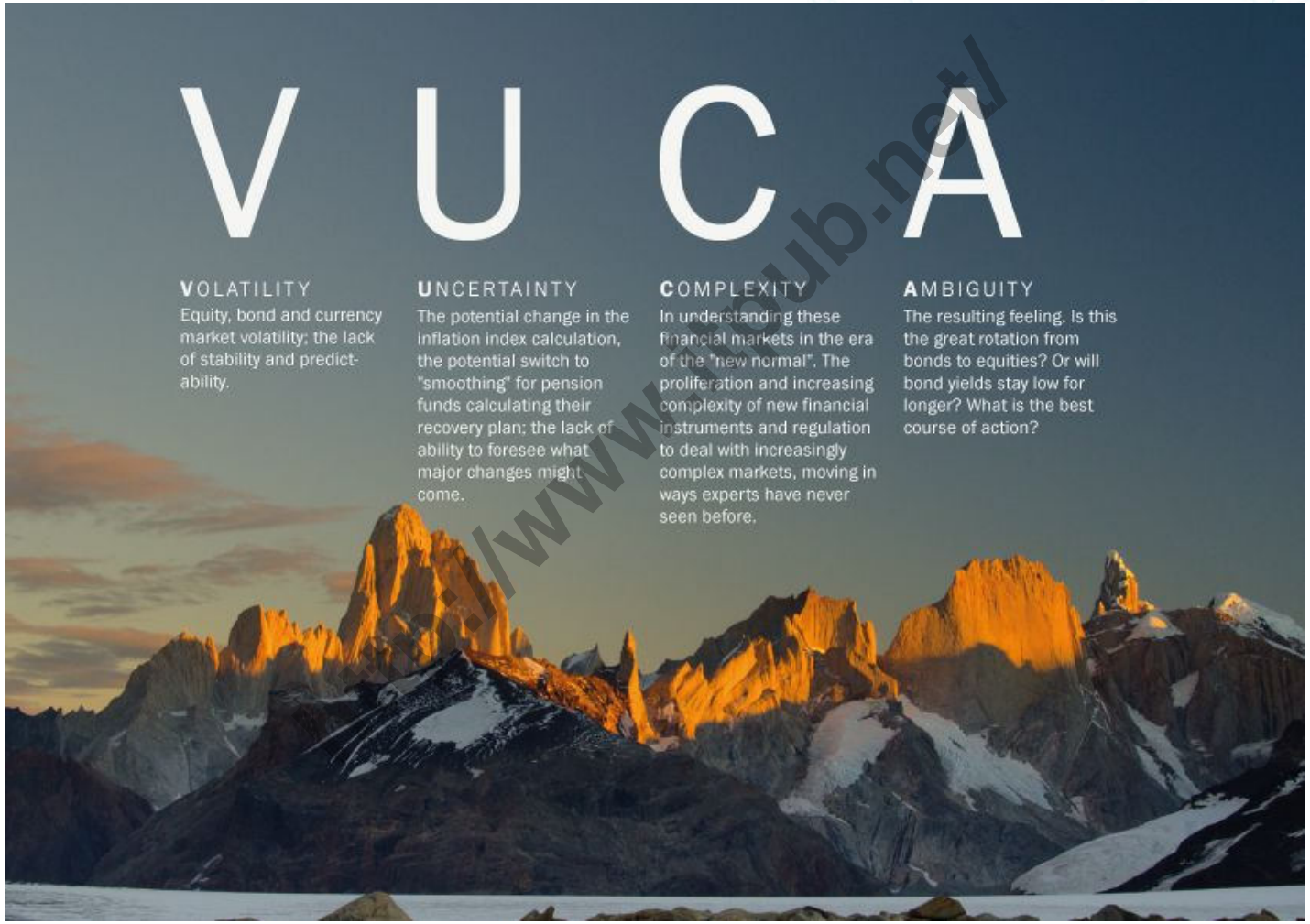


通用玩法 | 工具

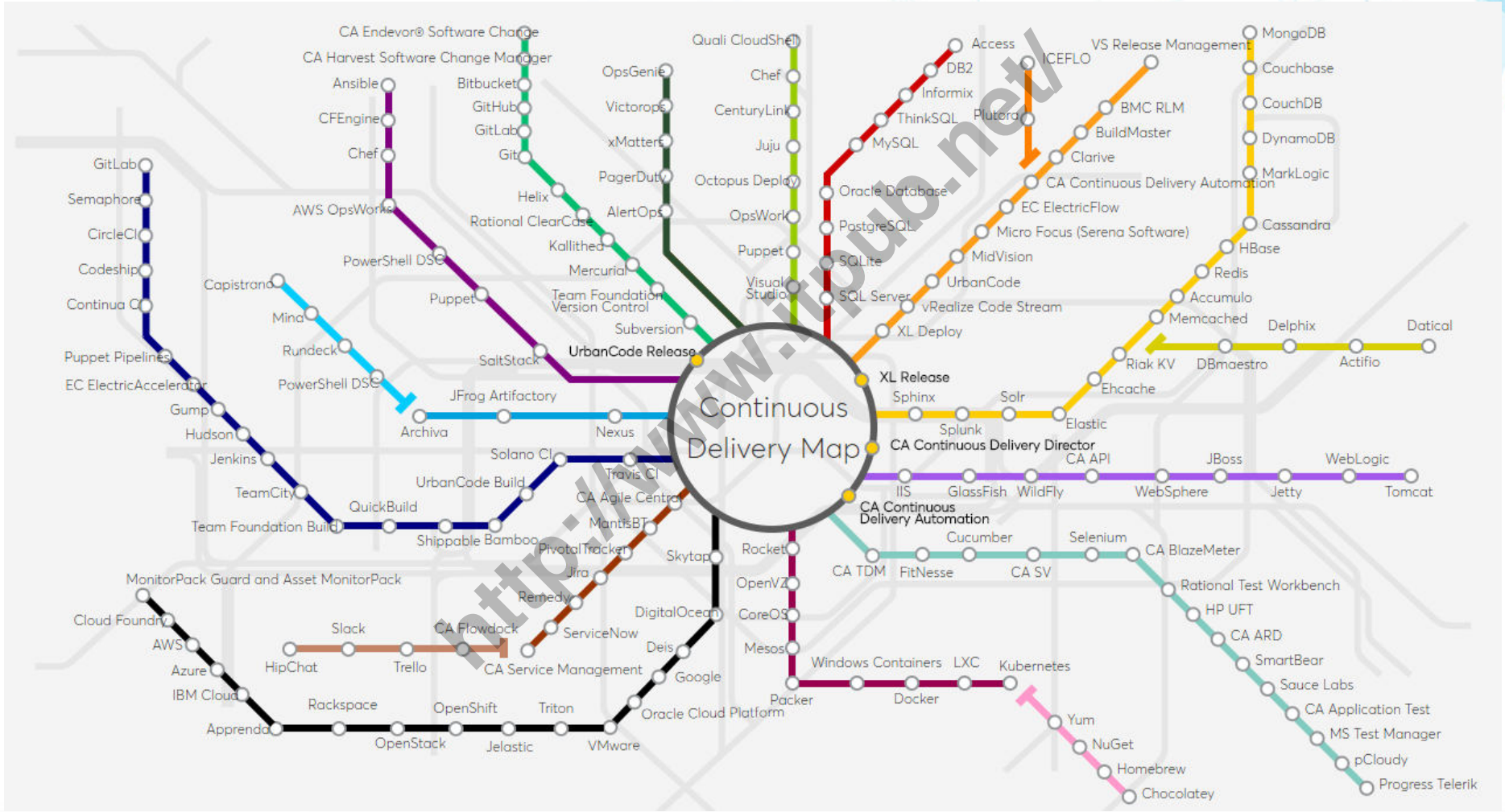


- 什么是云原生
- 云原生业界通用DevOps玩法（行业对标）
- AI领域在DevOps错与对和软着陆的抓手
- 云原生容器化DevOps流水线在AI领域的实战

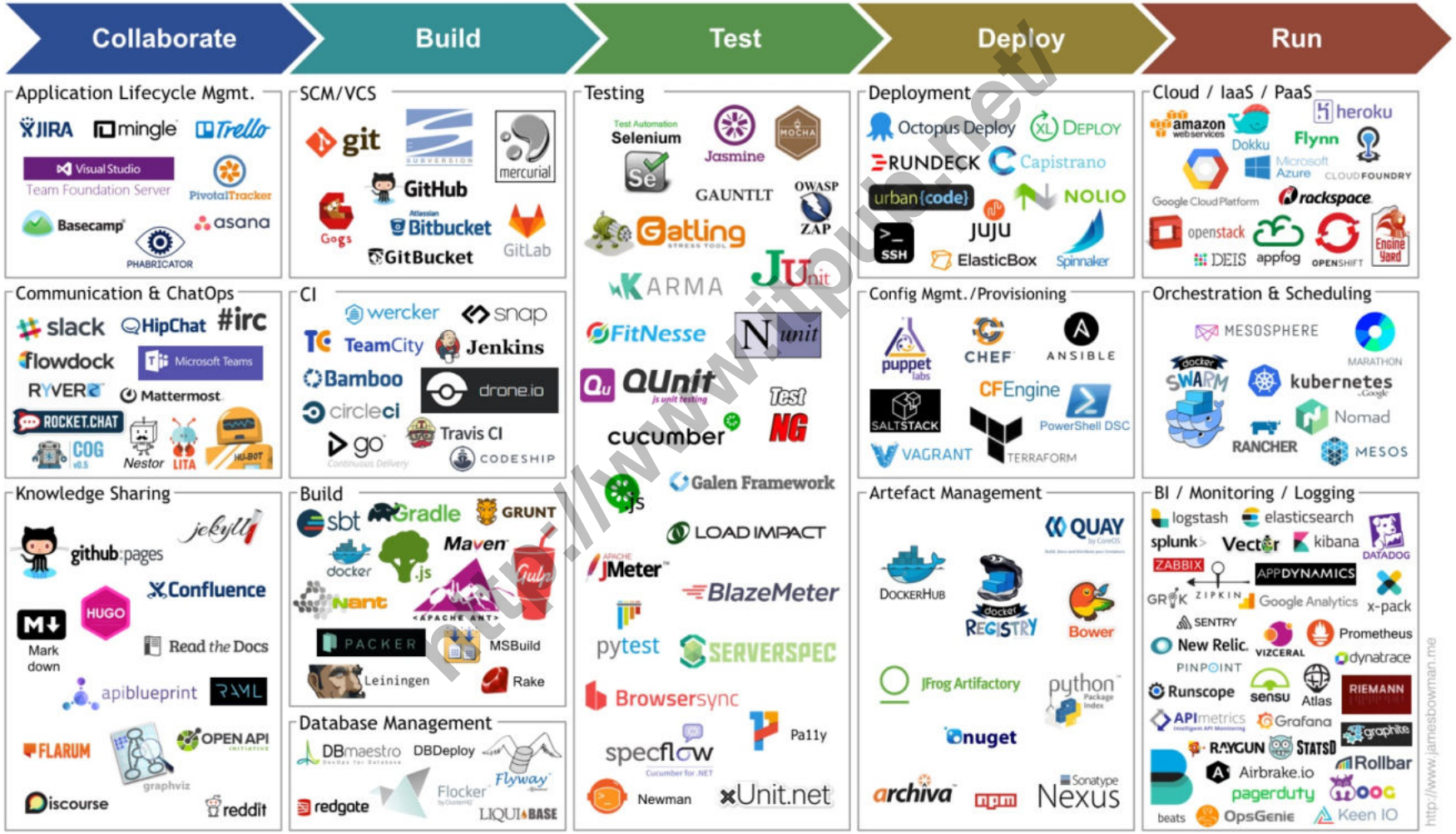
乌卡时代



Continuous Delivery Map



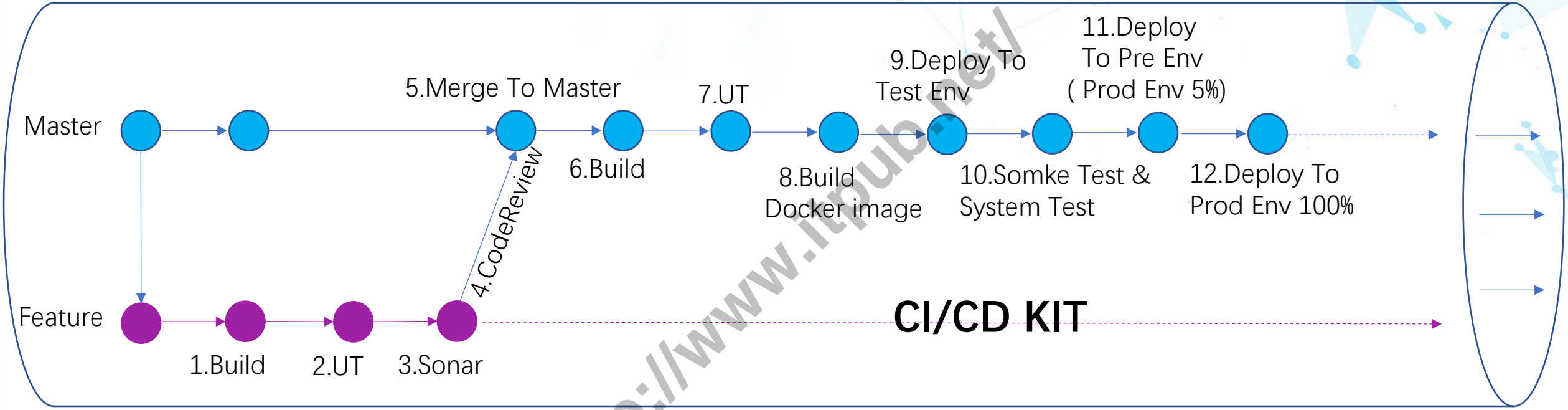
Continuous Delivery Tool Landscape



那些年我们使用过的 “法宝”

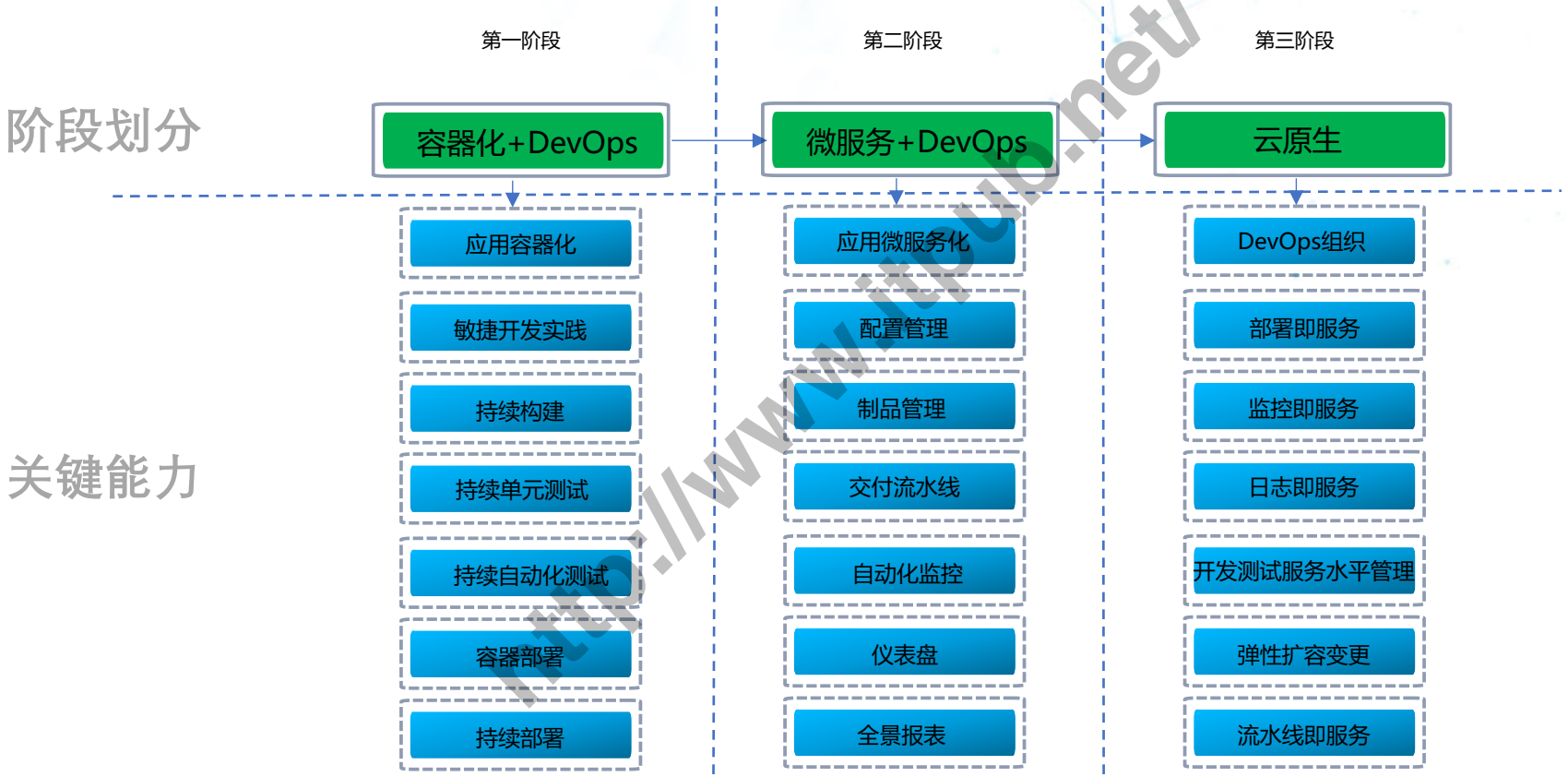
粒度 & 解耦

那些年我们使用过的 “法宝”



代码管理	代码扫描	持续集成	制品库	自动化测试	持续部署	持续监控、分析
	sonarqube			JUnit Se	docker kubernetes	Prometheus elastic

云原生DevOps天梯



云原生DevOps抓手

走出第一步

第一应用容器化

快速得到一个效果

- 如何选择单体应用
- 如何快速迁移上云
- 快速见到第一个效果

第一团队DevOps化

寻找精英团队

- 就像战场上的精英兵团一样，公司也需要精英团队
- 找到他们
- 给他们武器和鼓励

第一条工具链打造

工具孤岛≠工具链

- 单个工具是不成体系的，工具也需要集团作战
- 选择成熟度最高的工具
- 但不要停下调研最新工具的脚步

再好的理念，没有好的工具链支持，最终会因为实现成本昂贵而导致落地难！

不积跬步无以至千里

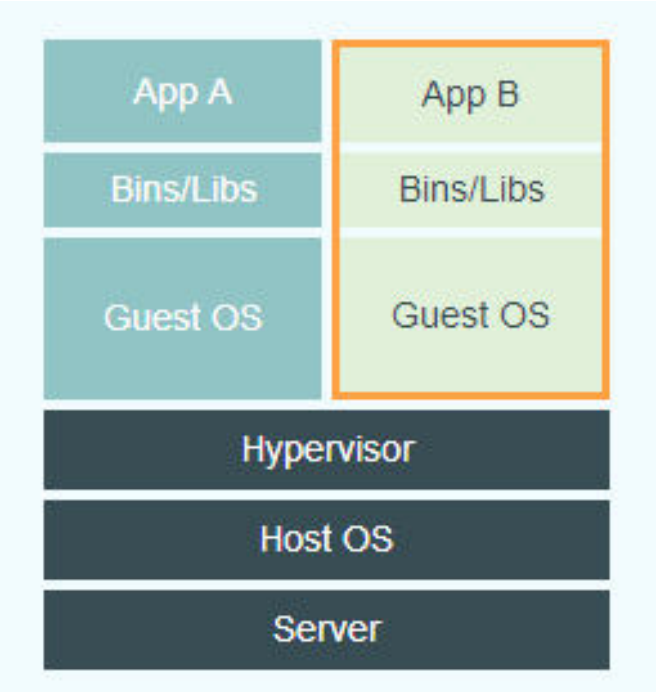
- 什么是云原生
- 云原生业界通用DevOps玩法（行业对标）
- AI领域在DevOps错与对和软着陆的抓手
- 云原生容器化DevOps流水线在AI领域的实战

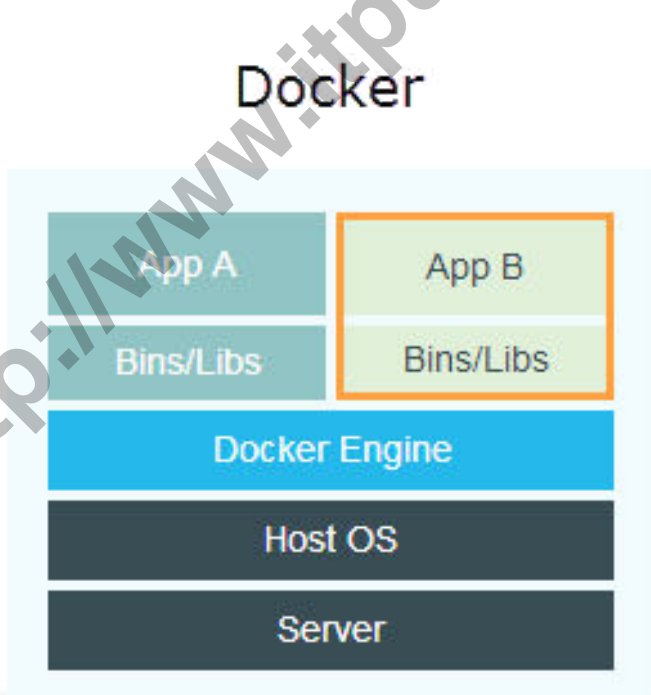
从容器化开始（最好的载体）

- 容器运行时 – docker rkt gvisor containerd…
- 容器编排系统 - kubernetes

定义

- 容器技术是基于linux内核的namespace隔离与cgroup资源配额技术实现的轻量级虚拟化技术，docker是其中一种实现





Docker核心概念 - 镜像与容器

- 镜像：一堆文件的集合以分层的方式存在机器上
- 容器：本质是一个隔离的运行环境
- 仓库：集中存放镜像的地方

- 通俗的类比：

	镜像	容器
类比操作系统	如同操作系统iso文件	如同一个装好的系统，启停容器如同开关机
类比编程	类	对象

Docker镜像从何而来

- Dockerfile
- cccp.yaml
- centos-7-docker.tar.xz
- centos-7-docker.tar.xz.asc

```
dev     mnt     srv
etc     opt     sys
home    proc    tmp
lib     root    usr
lib64   run     var
media   sbin
```

```
1   FROM scratch
2   ADD centos-7-docker.tar.xz /
3
4   LABEL org.label-schema.schema-version="1.0" \
5       org.label-schema.name="CentOS Base Image" \
6       org.label-schema.vendor="CentOS" \
7       org.label-schema.license="GPLv2" \
8       org.label-schema.build-date="20181006"
9
10  CMD ["/bin/bash"]
```


docker build –t centos:7 .

Dockerfile指令

Dockerfile 指令

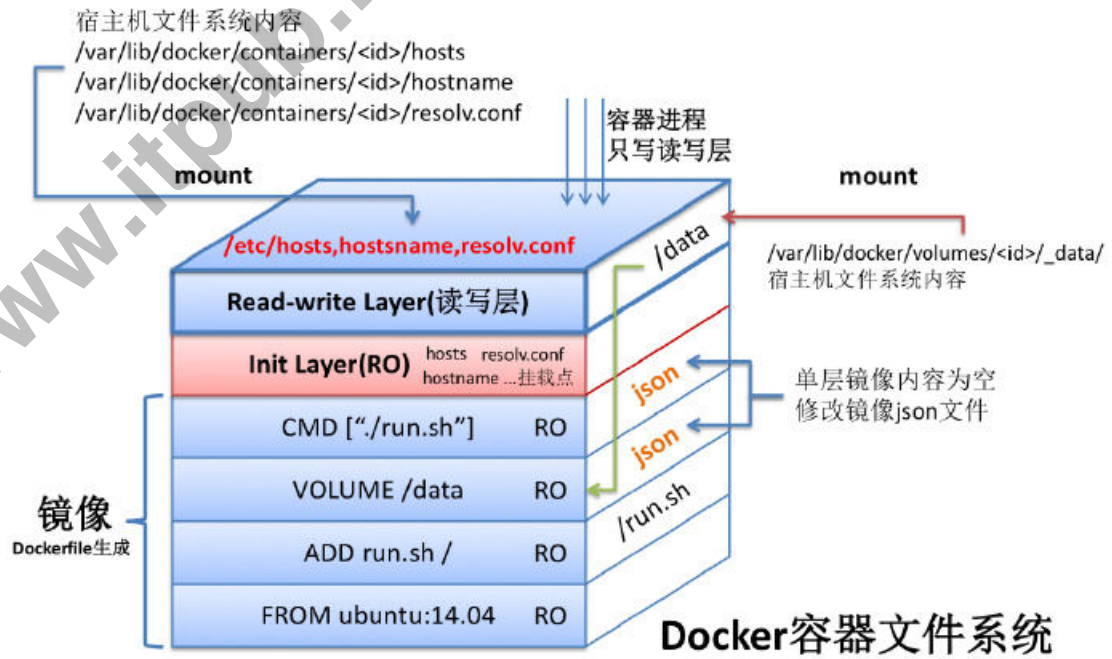
- COPY 复制文件
- ADD 更高级的复制文件
- CMD 容器启动命令
- ENTRYPOINT 入口点
- ENV 设置环境变量
- ARG 构建参数
- VOLUME 定义匿名卷
- EXPOSE 暴露端口
- WORKDIR 指定工作目录
- USER 指定当前用户
- HEALTHCHECK 健康检查

镜像分层原理 - 写时复制

- 构建镜像-Dockerfile

```
# cat Dockerfile
FROM ubuntu:14.04
ADD run.sh
VOLUME /data
CMD [ “./run.sh” ]
```

- docker build -t app:v3 .

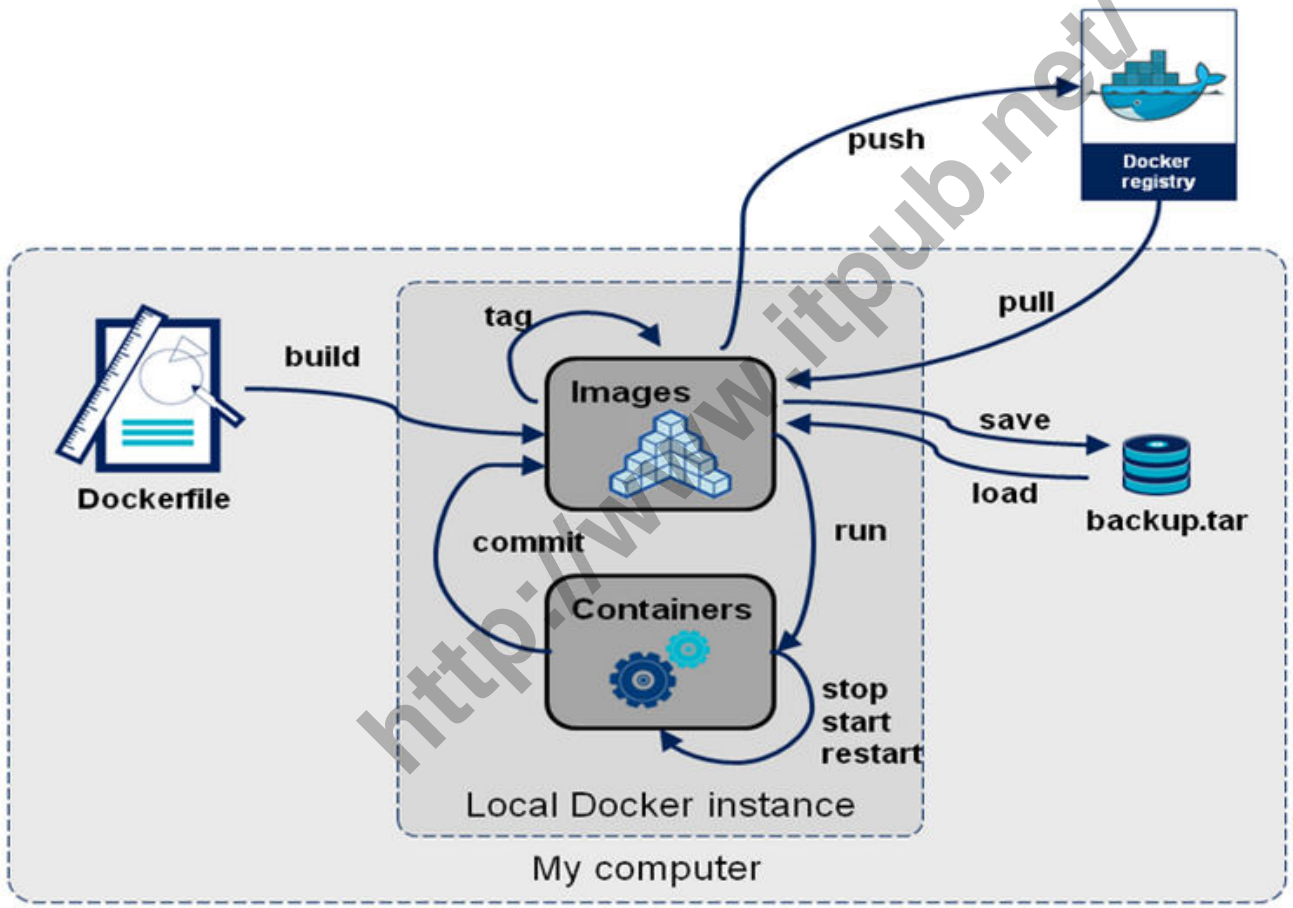


镜像原则

- 能轻不重
- 单一功能
- 合理分层
  - 层数
  - 复用
  - 顺序 - 经常变动放最上面

```
FROM centos:7.4.1708
RUN yum install -y  wget  \
    && yum install -y gcc-c++ openssl-devel openssl \
    && yum install -y net-tools
RUN wget http://www.keepalived.org/software/keepalived-2.0.8.tar.gz \
    && tar zxvf keepalived-2.0.8.tar.gz \
    && cd keepalived-2.0.8 && ./configure && make && make install
CMD keepalived --all -d -D  -f /etc/keepalived/keepalived.conf \
   --log-console --dont-fork
```

常用操作



容器原理

隔离-Namespace

分类	系统调用参数	相关内核版本
Mount namespaces	CLONE_NEWNS	Linux 2.4.19
UTS namespaces	CLONE_NEWUTS	Linux 2.6.19
IPC namespaces	CLONE_NEWIPC	Linux 2.6.19
PID namespaces	CLONE_NEWPID	Linux 2.6.24
Network namespaces	CLONE_NEWNET	始于Linux 2.6.24 完成于 Linux 2.6.29
User namespaces	CLONE_NEWUSER	始于 Linux 2.6.23 完成于 Linux 3.8)

主要涉及到三个系统调用:

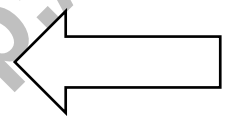
- **clone()** – 实现线程的系统调用，用来创建一个新的进程，并可以通过设计上述参数达到隔离。
- **unshare()** – 使某进程脱离某个namespace
- **setns()** – 把某进程加入到某个namespace

容器原理

隔离-Namespace

- pid 命名空间
  - 进程编号隔离

```
lwhong@ubuntu:~$ sudo ./pid
Parent [ 3474] - start a container!
Container [    1] - inside the container!
root@container:~# echo $$
1
```

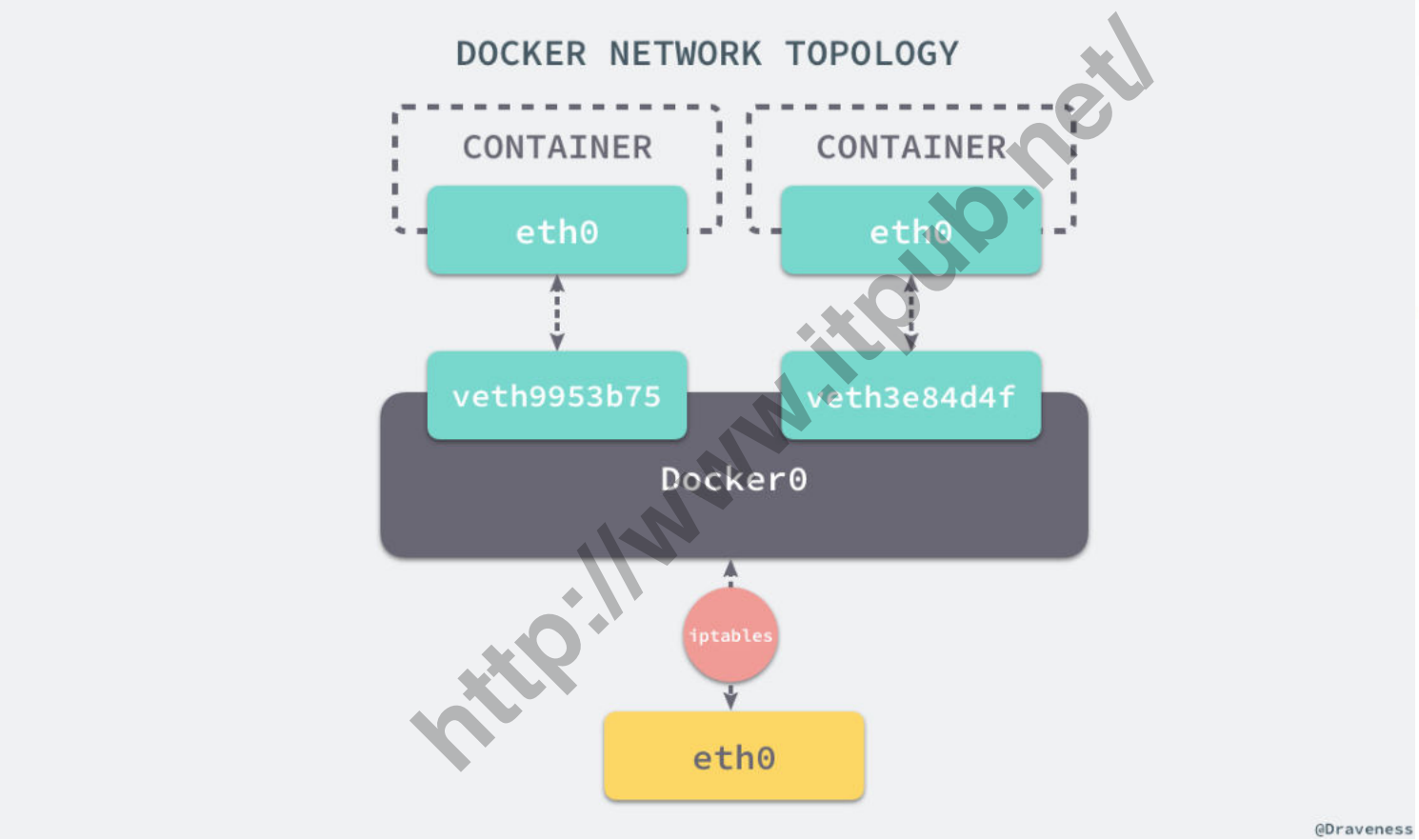


```
int container_main(void* arg)
{
    /* 查看子进程的PID，我们可以看到其输出子进程的 pid 为 1 */
    printf("Container [%5d] - inside the container!\n", getpid());
    sethostname("container",10);
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}

int main()
{
    printf("Parent [%5d] - start a container!\n", getpid());
    /*启用PID namespace - CLONE_NEWPID*/
    int container_pid = clone(container_main,
container_stack+STACK_SIZE,
            CLONE_NEWUTS | CLONE_NEWPID | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

常用操作

- 运行容器
  - docker run ubuntu:14.04 /bin/echo 'Hello world'
  - docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
  - docker run -v /data:/host/data -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
  - docker run -d -p 80:80 nginx
  - docker run --net=host -d nginx
- 查看容器列表
  - docker ps
- 查看容器详情
  - docker inspect [选项] <容器|镜像>

容器网络 – docker0

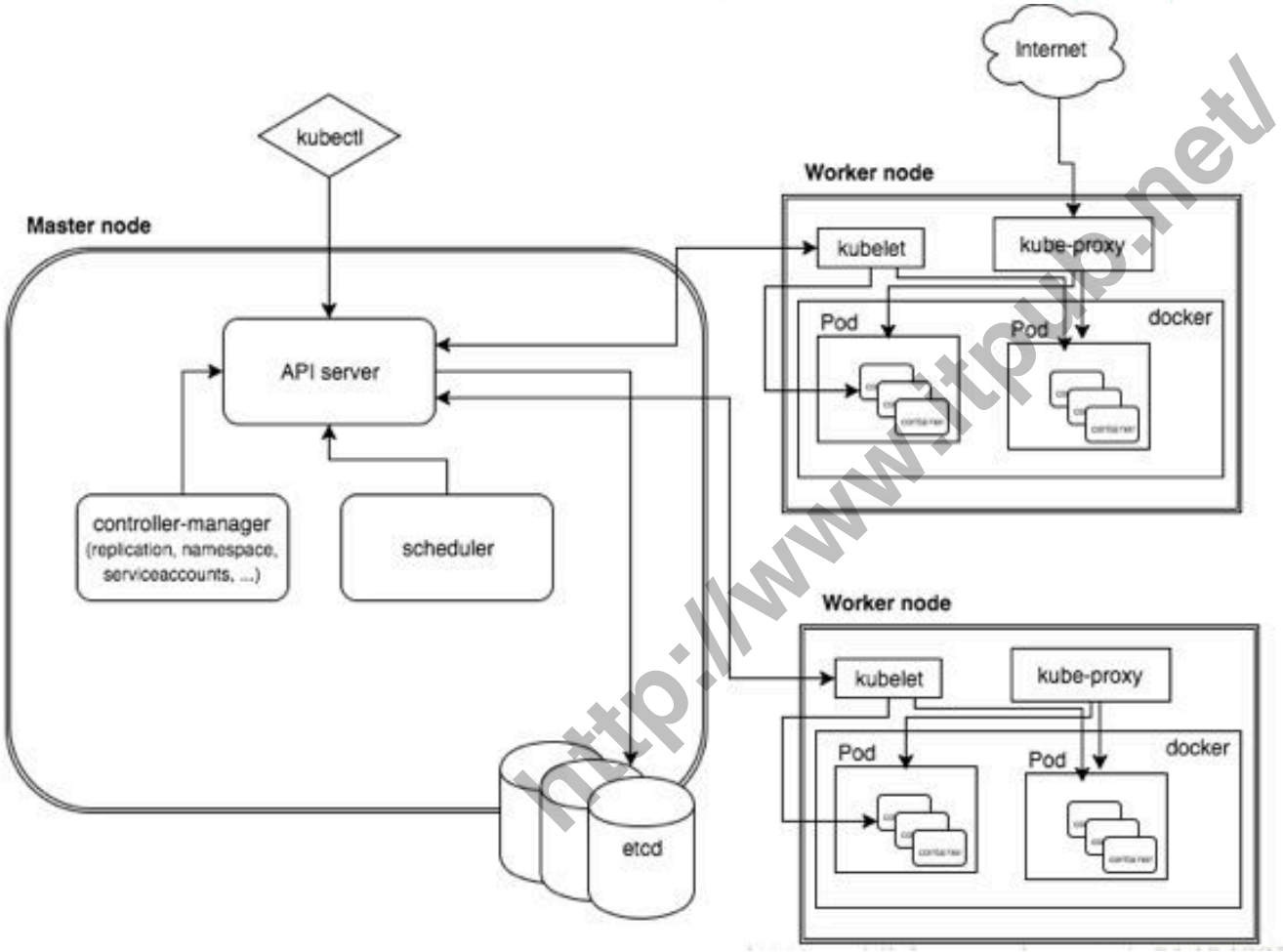


Docker进程模型

Code

```
docker      ctr
  |          |
  V          V
dockerd -> containerd ---> shim -> runc -> runc init -> process
                      |--> shim -> runc -> runc init -> process
                      |--> shim -> runc -> runc init -> process
```

Kubernetes介绍



容器编排调度系统

服务发现与负载均衡

自动打包

存储编排

故障自愈

滚动升级与回滚

配置与密钥管理

水平伸缩

基本概念 - Pod

Kubernetes最小调度单元，是多个容器的集合，pod内容器共享网络等一些namespace，像是个逻辑虚拟机

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

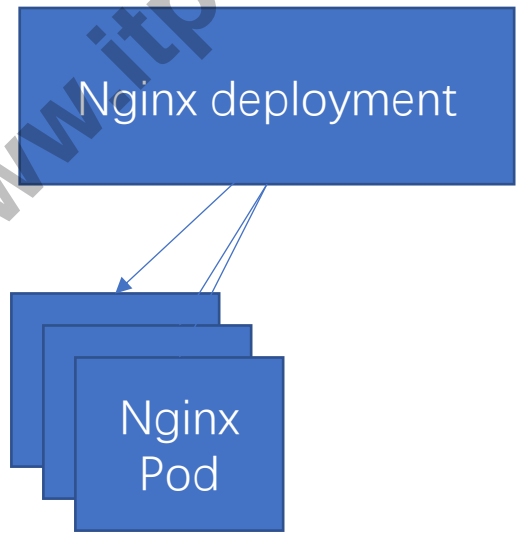
kubectl create –f pod.yaml

Kubernets中有很多“对象”
如pod就是一个，用一个“申明式”的yaml配置去描述这个“对象”

应用管理 - deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

kubectl create –f nginx.yaml



Pod的控制器，如保障副本数量
重启，滚动升级等

配置管理

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        # Provide the name of the ConfigMap containing the files you want
        # to add to the container
        name: special-config
  restartPolicy: Never
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.level: very
  special.type: charm
```

```
special.level
special.type
```

Define readiness probes

```
readinessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
livenessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 15
  periodSeconds: 20
```

tion might need to load large
ition, but you don't want to
se situations. A pod with
Services.

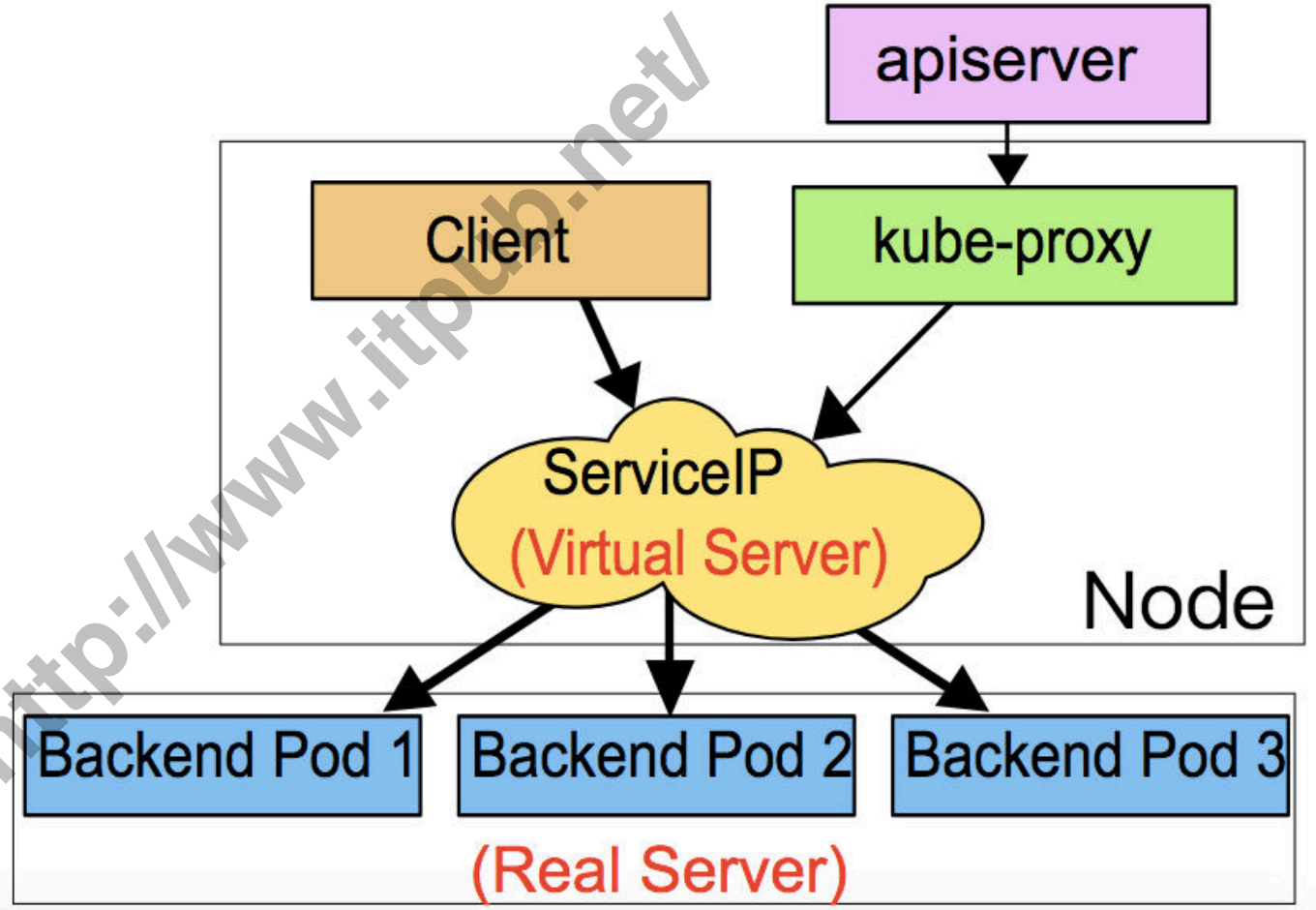
you use the

基本使用 – 负载均衡与服务发现

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

寻找贴有此标

nslookup my-servic



LB原理 – ipvs （负载均衡虚拟ip）

```
[root@dev-86-203 ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.1.86.203:32000 rr
  -> 100.66.33.196:8443           Masq    1      0          0
```

```
[root@dev-86-203 ~]# kubectl get pod -n kube-system -o wide|grep 196
kubernetes-dashboard-6948bdb78-cpt54       1/1       Running   0          3d        100.66.33.196   dev-86-203
```

基本使用 – 资源限制

```yaml
kind: Deployment
metadata:
  name: aaa
spec:
  replicas: 1
  selector:
    matchLabels:
      name: aaa
  template:
    metadata:
      labels:
        name: aaa
    spec:
      containers:
      - name: aaa
        image: busybox
        imagePullPolicy: Always
        command: ["sleep"]
        args: ["1000"]
        ports:
        - containerPort: 22
        resources:
          limits:
            memory: "1Mi"
            cpu: "1000m"
```

Resources.imits里进行限制

网络原理 – calico (容器虚拟ip)

```
[root@dev-86-203 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.1.86.1       0.0.0.0         UG    100    0        0 eth0
10.1.86.0       0.0.0.0         255.255.255.0   U     100    0        0 eth0
100.66.33.192   0.0.0.0         255.255.255.192 U     0      0        0 *
100.66.33.193   0.0.0.0         255.255.255.255 UH    0      0        0 cali2c6213a3c5e
100.66.33.194   0.0.0.0         255.255.255.255 UH    0      0        0 cali05022ccffca
100.66.33.195   0.0.0.0         255.255.255.255 UH    0      0        0 cali8b4137b517a
100.66.33.196   0.0.0.0         255.255.255.255 UH    0      0        0 cali1df5364df14
100.66.33.197   0.0.0.0         255.255.255.255 UH    0      0        0 calic84a0080ee6
100.66.33.198   0.0.0.0         255.255.255.255 UH    0      0        0 califd3b1eca2a1
100.77.83.128   10.1.86.204     255.255.25
100.93.26.0     10.1.86.202     255.255.25
100.102.105.128 10.1.86.205     255.255.25
172.17.0.0      0.0.0.0         255.255.0
```


```
[root@dev-86-202 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.1.86.1       0.0.0.0         UG    100    0        0 eth0
10.1.86.0       0.0.0.0         255.255.255.0   U     100    0        0 eth0
100.66.33.192   10.1.86.203     255.255.255.192 UG    0      0        0 tunl0
100.77.83.128   10.1.86.204     255.255.255.192 UG    0      0        0 tunl0
100.93.26.0     0.0.0.0         255.255.255.192 U     0      0        0 *
100.102.105.128 10.1.86.205     255.255.255.192 UG    0      0        0 tunl0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 br-011a84e735b7
172.19.0.0      0.0.0.0         255.255.0.0     U     0      0        0 br-91de43558bb5
```

理念

One Container One Process!!!

Using Dockerfile to Build Image

Container Read Only

容器技术的优势

标准化

一致性

隔离性

- 减少应用开发成本
- 自动化运维与管理
- 提升资源利用率

标准化

没有容器标准，系统针对进程进行管理调度监控等将是何等复杂？

没有集装箱技术，吊机，轮船，卡车将如何协同运作？

容器标准

- 应用标准
  - 镜像打包标准
  - 生命周期管理标准
  - 监控标准
- 系统标准
  - 计算存储网络接口标准

一致性

传统环境

Zlib:1.2.7
Python:2.7
IP：1.2.3.4
Centos:7.2

Zlib:1.2.5
Python:3
IP:2.3.4.5
Ubuntu:16.04

Zlib:1.2.7
Python忘了装
IP:4.5.6.7
Centos:7.4

容器化环境

APP APP APP
container

开发环境

APP APP APP
container

测试环境

APP APP APP
container

生产环境

隔离性

减少各应用之间的相互影响

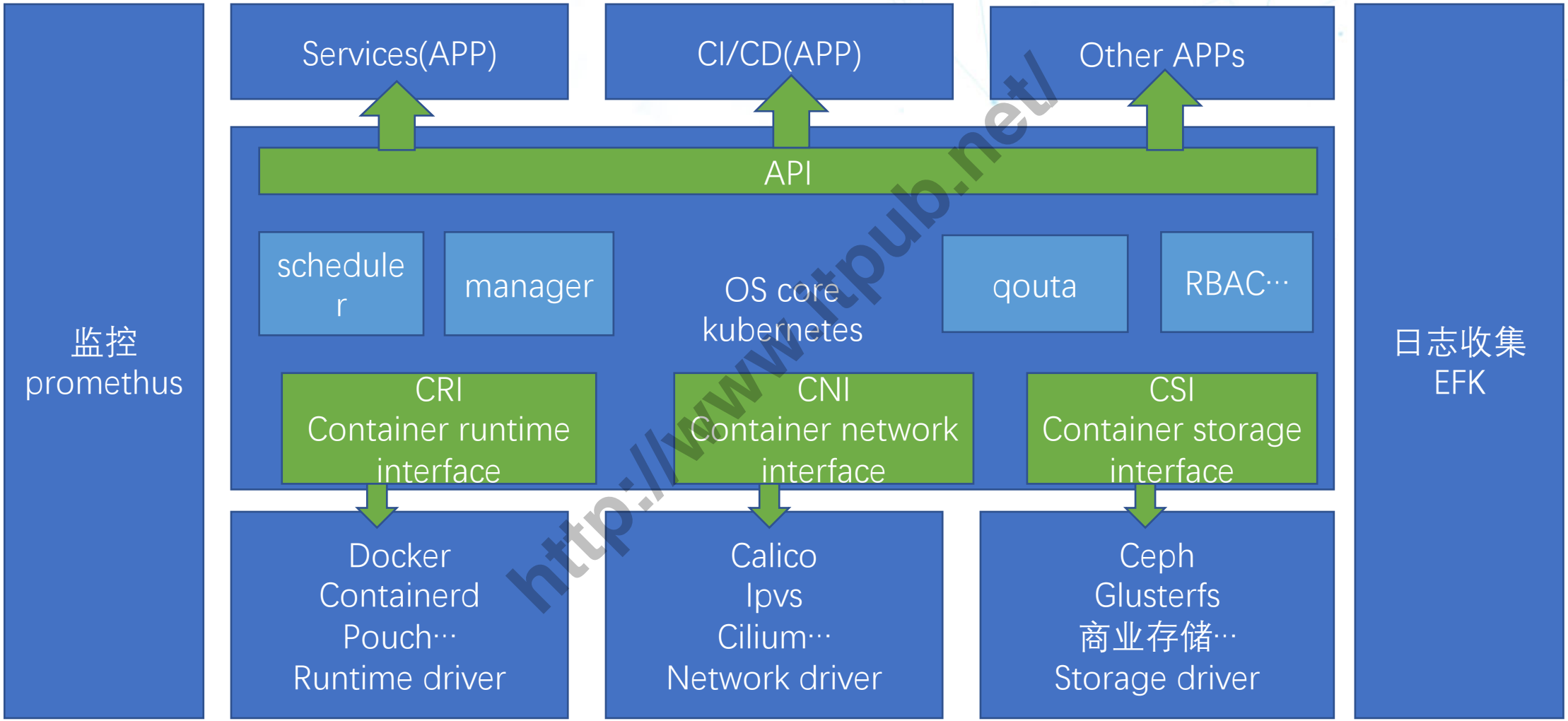
提升资源的复用性与利用率

更合理的去动态分配资源

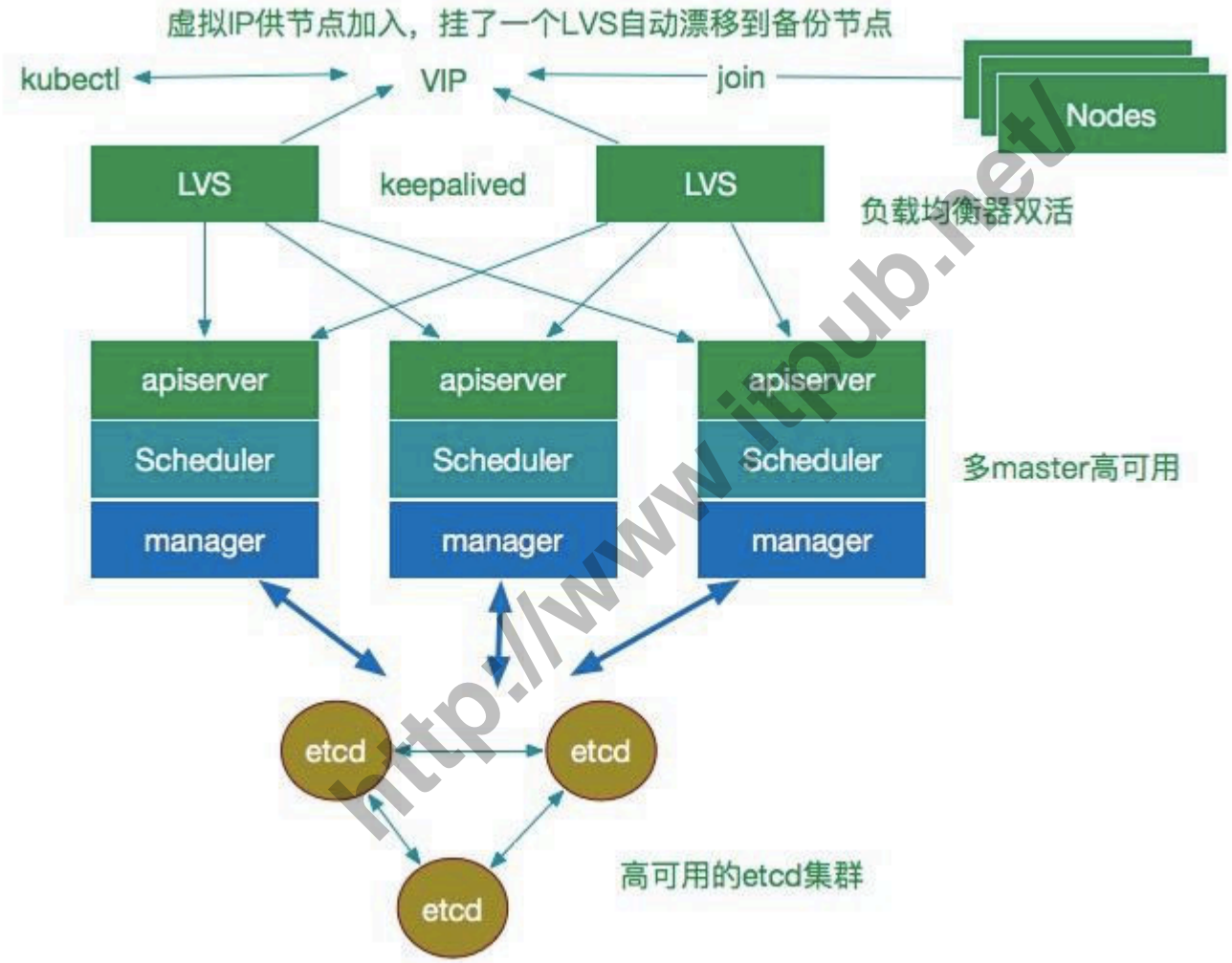
优势体现

	效果	举例
开发层面	充分利用平台自身特性去解决软件开发层面的问题	1. 利用kubernetes自带负载均衡与服务发现机制，业务层就可把这部分复杂逻辑剥离到系统层 2. 利用k8s的configmap去管理配置，无需管理配置中心这些逻辑，或者采用环境变量与命令行启动参数传输少量配置 3. 利用sidecar机制进行全链路跟踪，监控与业务层解耦合
软件交付层面day1	1. 代码定义流程，完全自动	1. 通过CI/CD系统明确定义整个软件交付的过程
应用生命周期管理day2	实现系统管理应用，而非人去管理	1. 使用编排文件告诉系统你的应用如何启动 2. 监控系统与应用之间实现解耦，使监控系统实现统一，应用自身编排配置包含监控和播测信息等，且容器生命周期与应用绑定，监控容器既是监控应用 3. 统一调度，故障自愈等
资源利用层面	提升资源复用 动态调度，资源超分 弹性计算	1. PaaS平台10个物理节点，部署的300多应用实例，还可进一步复用，这在传统架构下难以想象 2. 用户申请量高，实际使用量低对集群资源超分释放出100核以上的资源 3. 通过弹性计算技术可以根据业务量动态调整应用实例数

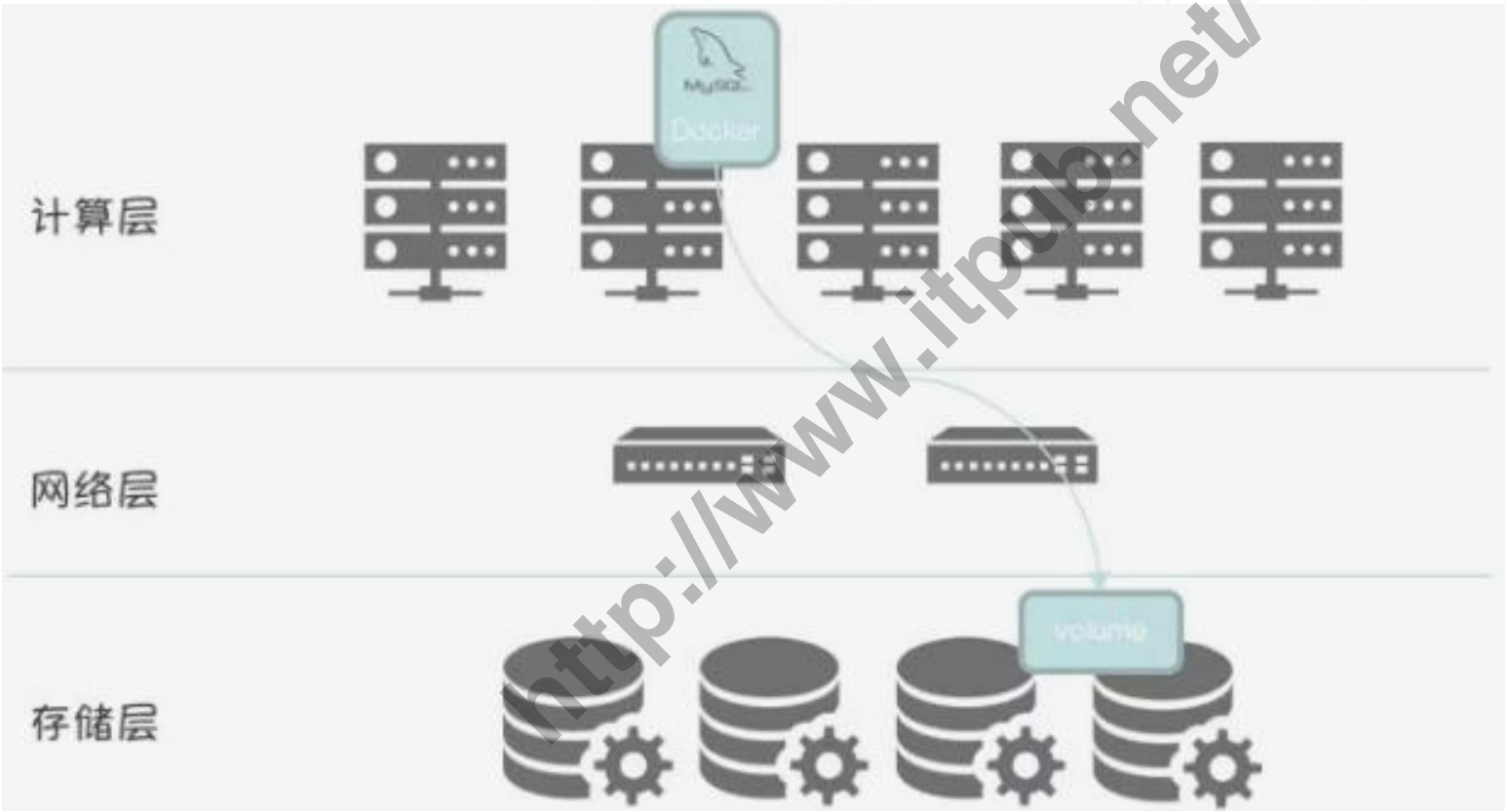
容器平台架构简图



核心组件高可用架构



计算存储分离



网络虚拟化

- 容器采用路由网络
- 出口实现lvs集群，等价路由协议
  - 无单点，多主，可横向扩展网络出口节点
- 网络TOA技术实现源地址保留

容器化阻碍

- 意识不到容器技术带来的好处
- 担心稳定性
- 业务人员不熟悉容器
- 担心改造成本过高

树立信心

- 容器技术现在业内大规模应用已经成熟
- 不用担心业务复杂上不了容器，容器就是为了解决这些问题的
  - Openstack这种超级复杂的应用平台都可以跑在容器上
- 担心改容器化造成本太高，业务人员不熟悉容器
  - 首先业务需要拥抱大势，工具层面会解决易用性问题
  - 该改造的还是需要改造，长痛不如短痛

理念

- 应用被设计成由软件和系统去管理的，而不是由人管理
- 把集群当成一个整体，而非单个主机
- Everything as code
  - 用代码定义运行环境，编译打包交付流程，运行方式，健康检测等
- 尽可能无状态
  - 利用分布式存储 分布式缓存数据库等处理状态
- 减少交互式操作

容器化改造

这块CI/CD工具可以协助解决，且主流编程语言都有模板可以使用

Dockerfile pipeline yaml配置谁写？

改变传统的交付与管理模式

- 镜像交付，明确定义交付流程（pipeline）
- 从进程纬度管理应用上升到容器纬度

利用容器系统自身特性解决应用问题

- 使用原生服务发现与负载均衡
  - 从业务逻辑中剥离这部分逻辑，用service做负载，DNS做服务发现
- 改变监控方式
  - 监控容器就是监控进程
  - 业务自身暴露metric
  - 使用播测
  - 使用sidecar机制做APM
- 管理配置
  - 少量配置使用环境变量或者命令行参数
  - 配置文件使用configmap,拒绝通过宿主机挂载配置文件
  - 热更新配置应用应该自身提供API，如同Envoy

结合CI/CD

Chart	e32eea9dbb	add changlog	1 周之
Godeps	accb498eb8	add pkg dependency	1 周之
client	5801d1c0a4	use incluster config	4 周之
cmd	f84c470db7	add ?scharset=utf8&parseTime=True&loc=Local	2 周之
db	4c0fc600bb	set connmaxlifetime	2 周之
docs	e32eea9dbb	add changlog	1 周之
flags	eb72da70a0	promise after code review2	1 月之
modules	1bffc51f22	[CI SKIP]	1 周之
server	7a2f39ffd0	change to mysql	2 周之
structs	a6893ee0b1	add deploment wrong message	3 周之
test	e32eea9dbb	add changlog	1 周之
utils	3e431ca2ba	delete some useless import package	1 周之
vendor	1ae3fff1dd	change app_test for validate false	1 周之
.drone.yml	1bffc51f22	[CI SKIP]	1 周之
.gitignore	43ccd07d05	add print kubeconfig	1 月之
CHANGELOG.md	e32eea9dbb	add changlog	1 周之
Dockerfile	d016e8254f	dockerfile	3 月之

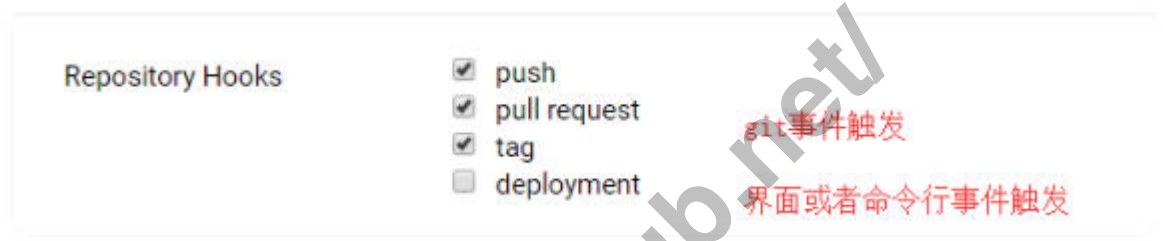
应用如何部署

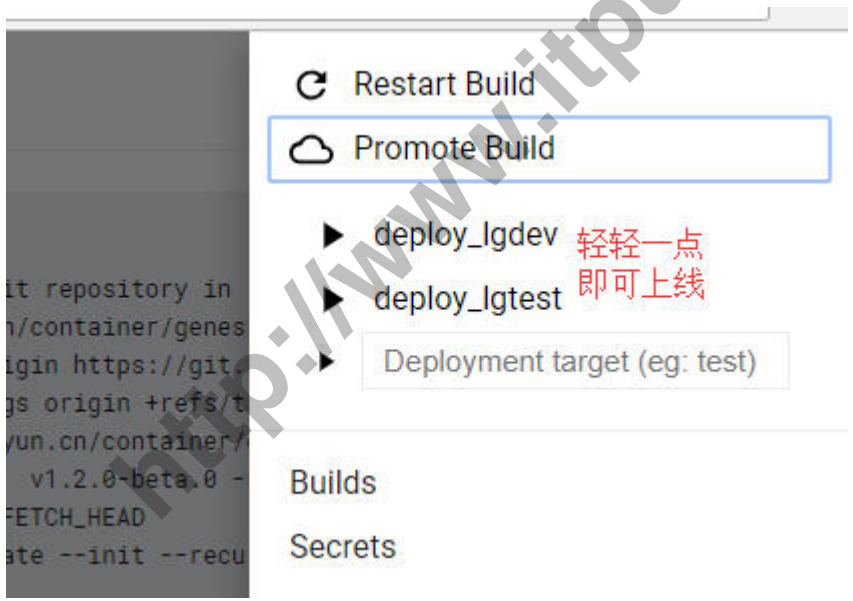
应用构建发布流程

应用如何构建

结合CI/CD

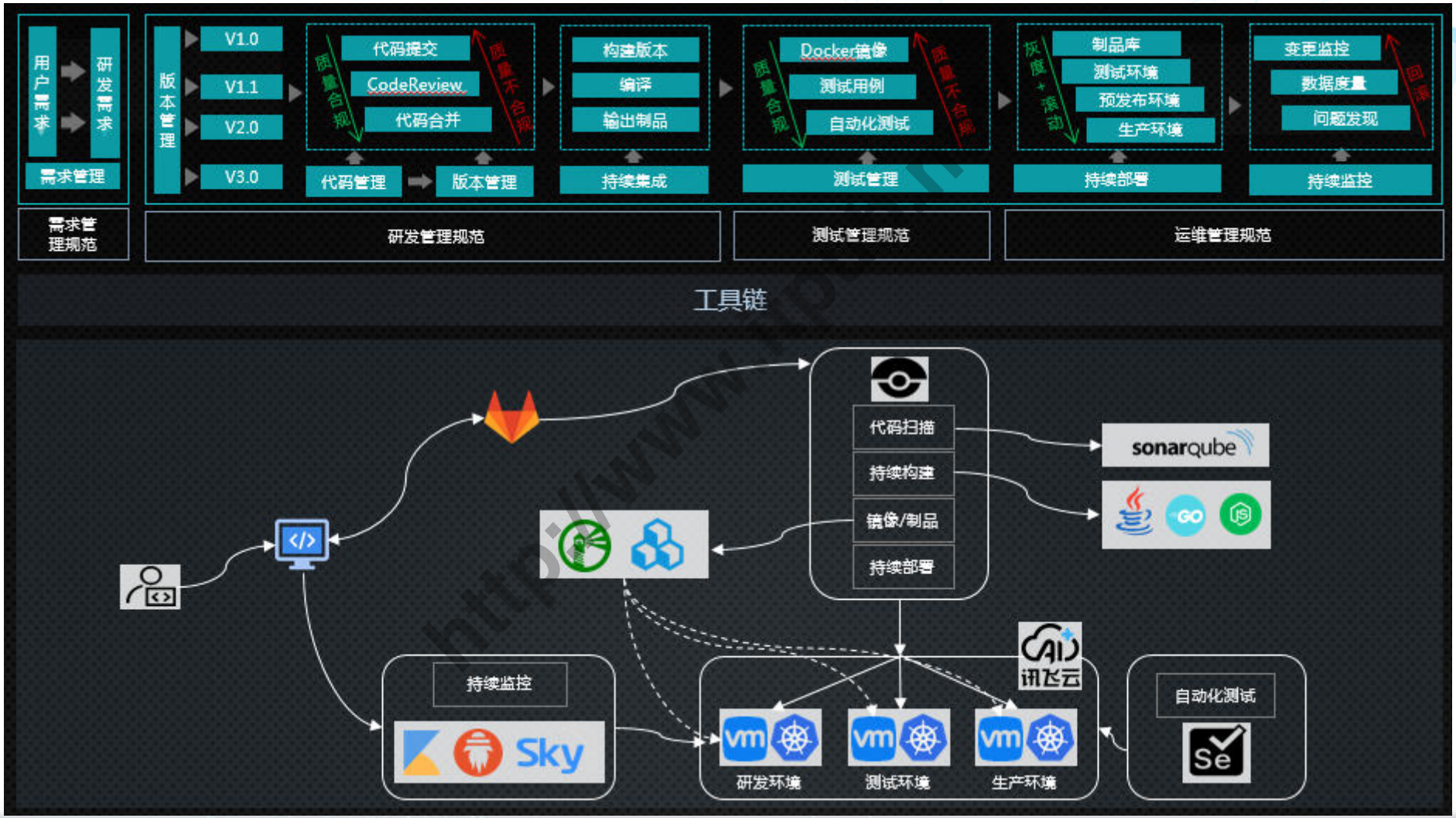
```
deploy_lgdev101.194: 测试环境
    image: 172.16.59.153/devops/helm:v2.8.1
    commands:
        - mkdir -p /root/.kube && cp -r .kube/co
        - helm delete genesis --purge || true
#        - helm repo add ifly http://172.16.59.1
        - helm install --name genesis  --set ima
    when:
        event: deployment通过deploy事件触发
        environment: deploy_lgdev

test_lgdev101.194:测试环境部署完跑测试用例
    image: 172.16.59.153/base/golang:1.8.0 # ven
    commands:
        - cd test
        - sleep 100
        -  go test -args --server "http://172.29
    when:
        event: deployment
        environment: deploy_lgdev

deploy_lgtest101.84:生产环境
    image: 172.16.59.153/devops/helm:v2.8.1
    commands:
        - mkdir -p /root/.kube && cp -r .kube/co
        - helm delete genesis --purge || true
#        - helm repo add ifly http://172.16.59.1
        - helm install --name genesis  --set ima
    when:
        event: deployment
        environment: deploy_lgtest

test-lgtest101.84:生产环境部署完跑测试用例
    image: 172.16.59.153/base/golang:1.8.0 # ver
```



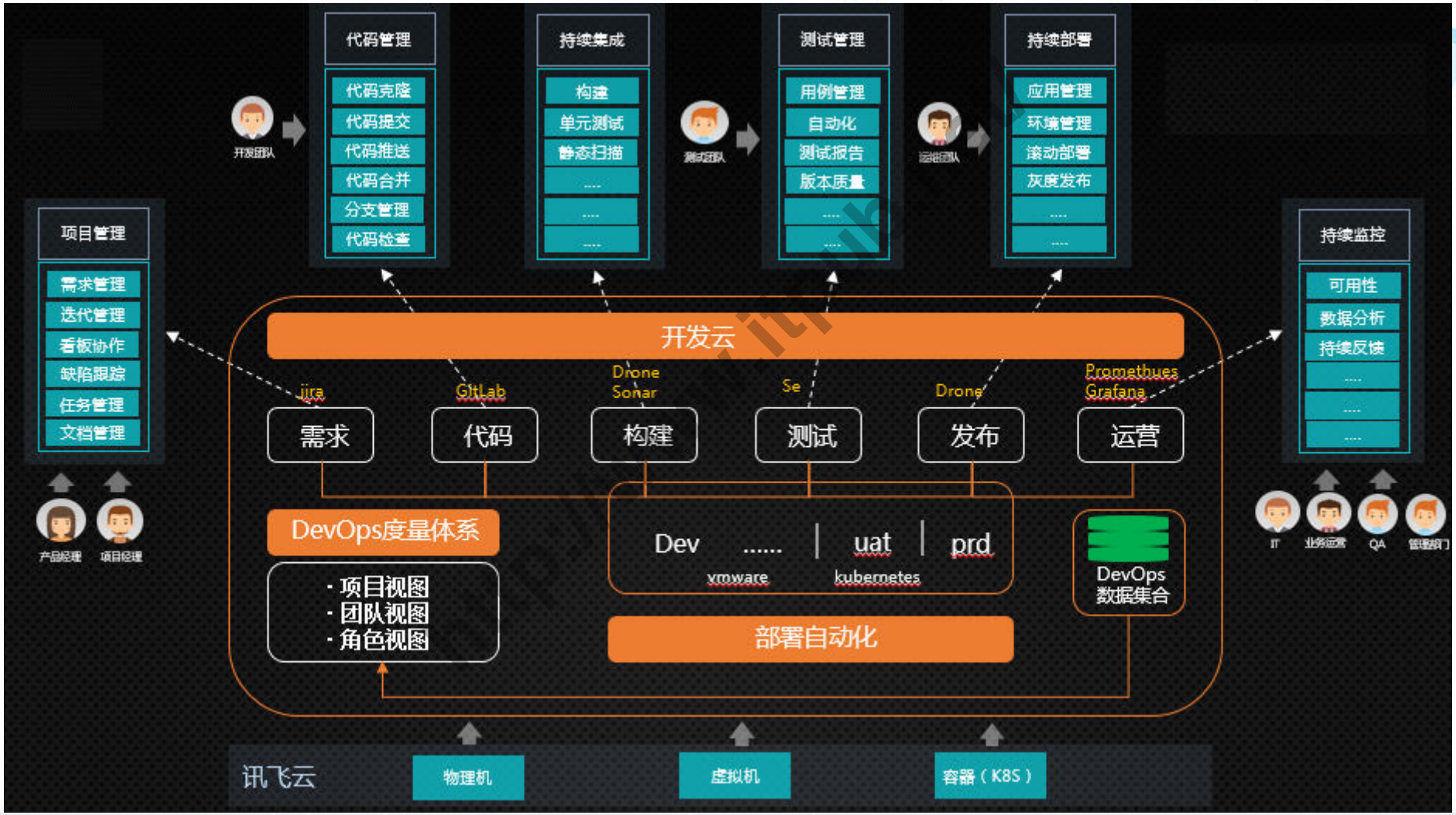


- git push 事件开发测试
- git tag 事件版本发布
- 手动触发上线部署

技术问题还是需要技术手段解决



从市场中来，到产品中去


代码管理
代码克隆
代码提交
代码推送
代码合并
分支管理
代码检查
持续集成
构建
单元测试
静态扫描
测试管理
用例管理
自动化
测试报告
版本质量
持续部署
应用管理
环境管理
滚动部署
灰度发布
项目管理
需求管理
迭代管理
看板协作
缺陷跟踪
任务管理
文档管理
持续监控
可用性
数据分析
持续反馈
开发团队
测试团队
运维团队
开发云
jira
GitLab
Drone
Sonar
Se
Drone
Promethues
Grafana
需求
代码
构建
测试
发布
运营
DevOps度量体系
· 项目视图
· 团队视图
· 角色视图
Dev
.....
vmware
uat
kubernetes
prd
部署自动化
DevOps
数据集合
产品经理
项目经理
IT
业务运营
QA
讯飞云
物理机
虚拟机
容器（K8S）

云原生容器化DevOps实战

SACC 2020
中国系统架构师大会
SYSTEM ARCHITECT CONFERENCE CHINA

DevOps Demo….

留给“我们”的时间不多了

10年前，云计算萌芽的时候，我们没抓住

5年前，Go语言快速发展，我们勉强跟上了节奏

现在，我们不能再错过云原生的革命了

Thanks