微服务自动伸缩

彭宇 @ douban

2017-09-23

背景

豆瓣简介和DAE

豆瓣简介

- 豆瓣于2005年3月 上线，是以技术和产品为核心、生活和文化为内容的创新网络服务

豆瓣读书　豆瓣电影　豆瓣音乐　豆瓣同城

豆瓣小组　豆瓣阅读　豆瓣FM douban.fm　豆瓣市集

1.3亿
注册用户

2亿
月活跃用户

2.4亿
日均PV

43分钟
登录用户平均停留时间

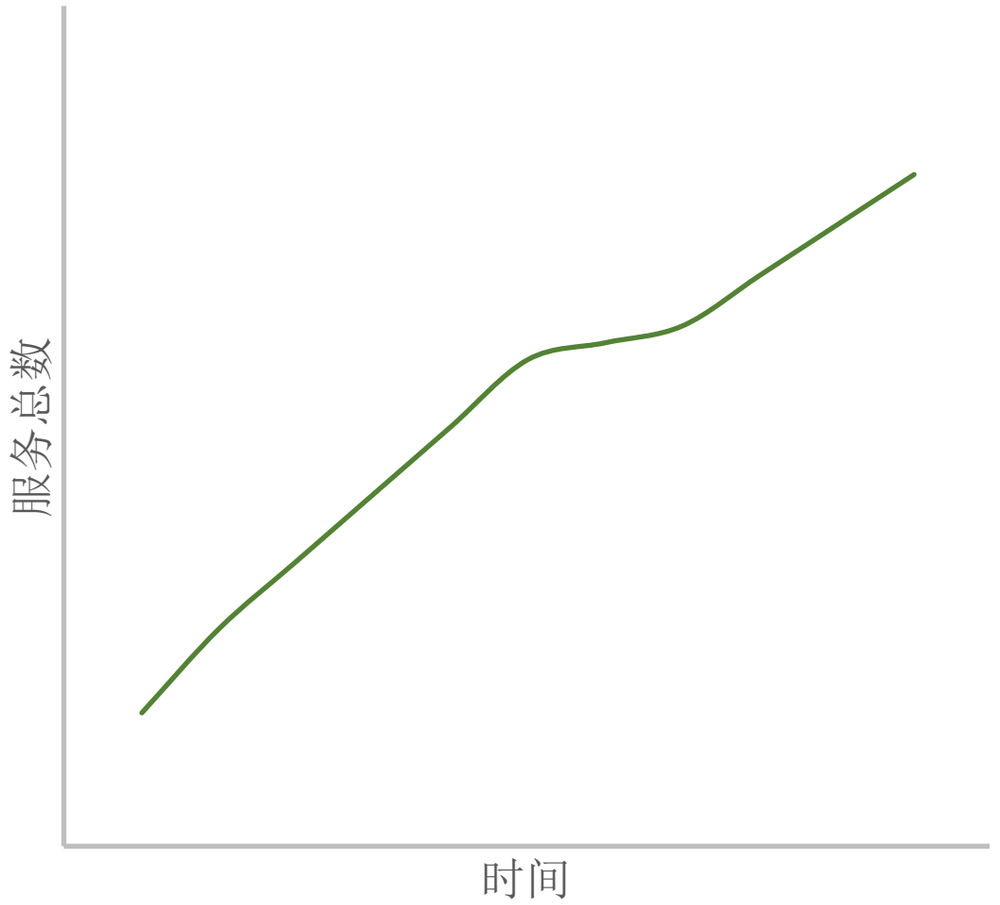
豆瓣 douban

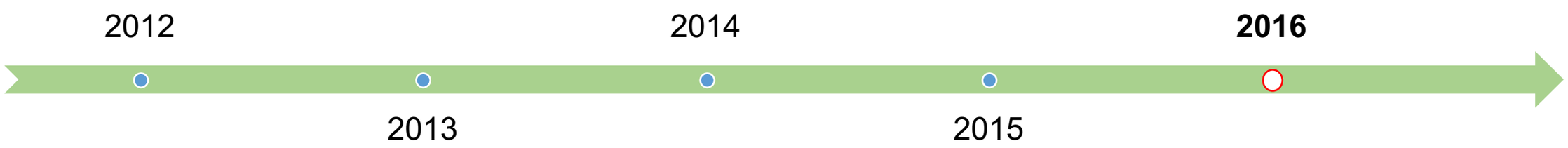
Douban App Engine

- 托管无状态应用 web/service（python／golang）
- 使用配置描述应用：服务依赖 / mq／daemon／cron
- 统一调度资源，在线worker和离线daemon
- 测试集成

现状

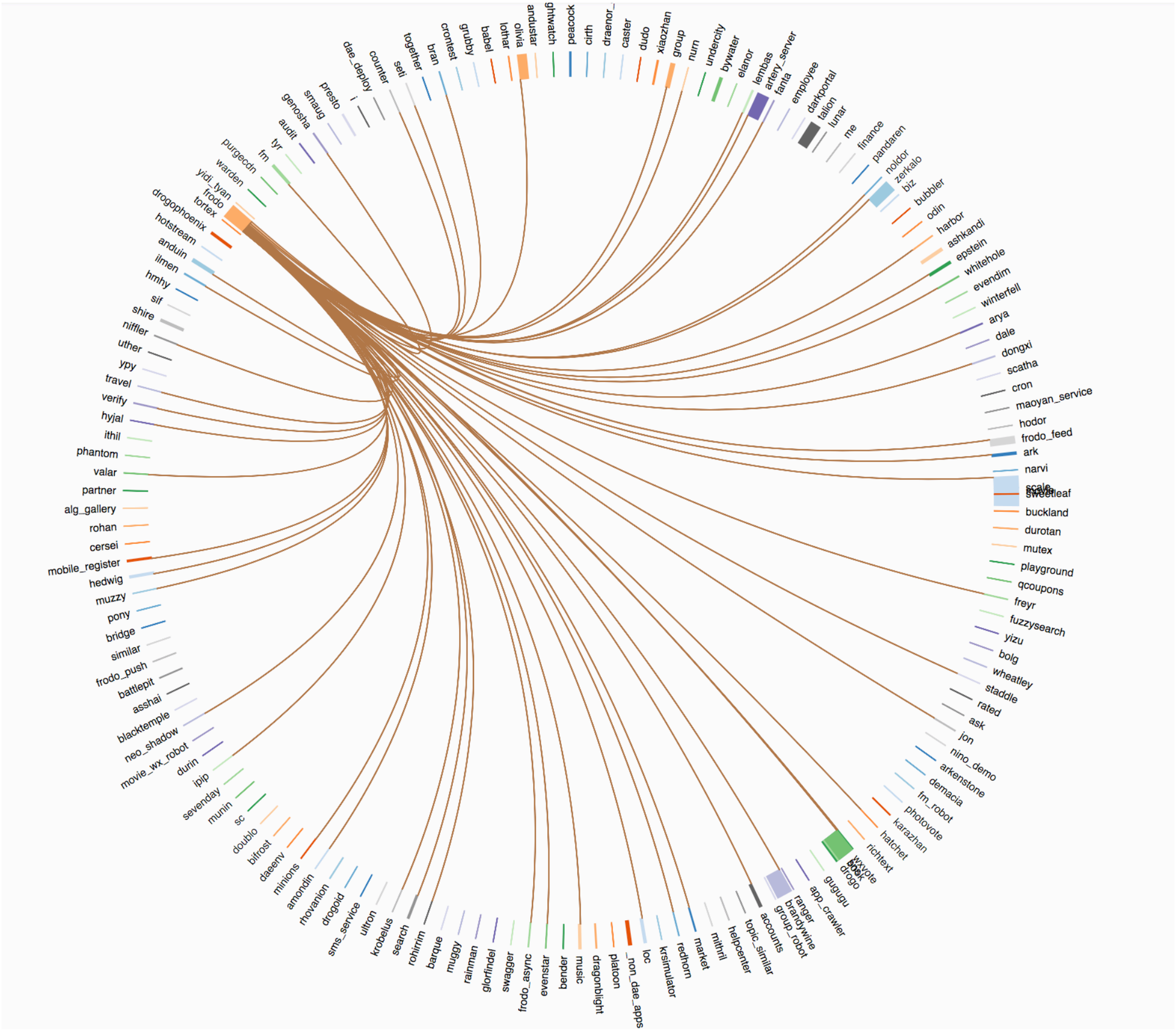
- 90%+ 的产品服务化
- 性能基线保持不变
- 全站可用性提升
- 每日发布变为随时发布





目标

为什么要做autoscale？


andustar
ghtwatch
peacock
cirth
draenor_
caster
dudo
xiaozhan
group
nurn
undercity
bywater
elanor
lembas
artery_server
fanta
employee
darkportal
talion
lunar
me
finance
pandaren
noldor
zerkalo
biz
bubbler
odin
harbor
ashkandi
epstein
whitehole
evendim
winterfell
arya
dale
dongxi
scatha
cron
maoyan_service
hodor
frodo_feed
ark
narvi
sweetleaf
buckland
durotan
mutex
playground
qcoupons
freyr
fuzzysearch
yizu
bolg
wheatley
staddle
rated
ask
jon
nino_demo
arkenstone
demacia
fm_robot
photovote
karazhan
hatchet
richtext
wwvote
drogo
gugugu
app_crawler
ranger
brandywine
group_robot
accounts
topic_similar
helpcenter
mithril
market
redhorn
krsimulator
loc
_non_dae_apps
platoon
dragonblight
music
bender
evenstar
frodo_async
swagger
glorfindel
rainman
muggy
barque
rohirrim
search
krobelus
ultron
sms_service
drogoid
rhovanion
amondin
minions
daeenv
bifrost
doublo
sc
munin
sevenday
ipip
durin
movie_wx_robot
neo_shadow
blacktemple
asshai
battlepit
frodo_push
similar
bridge
pony
muzzy
hedwig
mobile_register
cersei
rohan
alg_gallery
partner
valar
phantom
ithil
hyjal
verify
travel
ypy
uther
niffler
shire
sif
hmhy
ilmen
anduin
hotstream
drogophoenix
tortex
frodo
yidi_tyan
warden
purgecdn
fm
tyr
audit
genosha
smaug
presto
i
dae_deploy
counter
seti
together
bran
crontest
grubby
babel
lothar
olivia

服务化之前的做法

- SA到服务器上部署单体应用
- 按照服务器的处理能力设置相应的进程数
- 处理能力不足后加服务器

目标

- 提高服务器利用率，降低成本
- 提高运维自动化，减少人工重复性操作

方案

时间

- 2013.12 第一个测试应用
- 2014.2 对接线上应用
- 2014.3 接管dae上的所有应用 140+

  .docker还未引入 [2014Q4 DAE docker改造，2015Q2 完成docker化改造]

  .k8s还未发布
- 随着服务化的继续，持续优化和提高稳定性，目前管理500+应用

DAE scale 模型

- **应用:**
  - 一个git repo映射到DAE平台上的资源分配、调度、计费的逻辑集合
  - 全局唯一且相互独立，通过thrift/pidl/http RPC 相互调用
  - 对于web/thrift/pidl online service而言，支持multi-instances
  - 每个App资源分配不均等，不指定具体数字，只做范围限制，从workers实际使用看，最少2个，最多2800个

DAE scale 模型

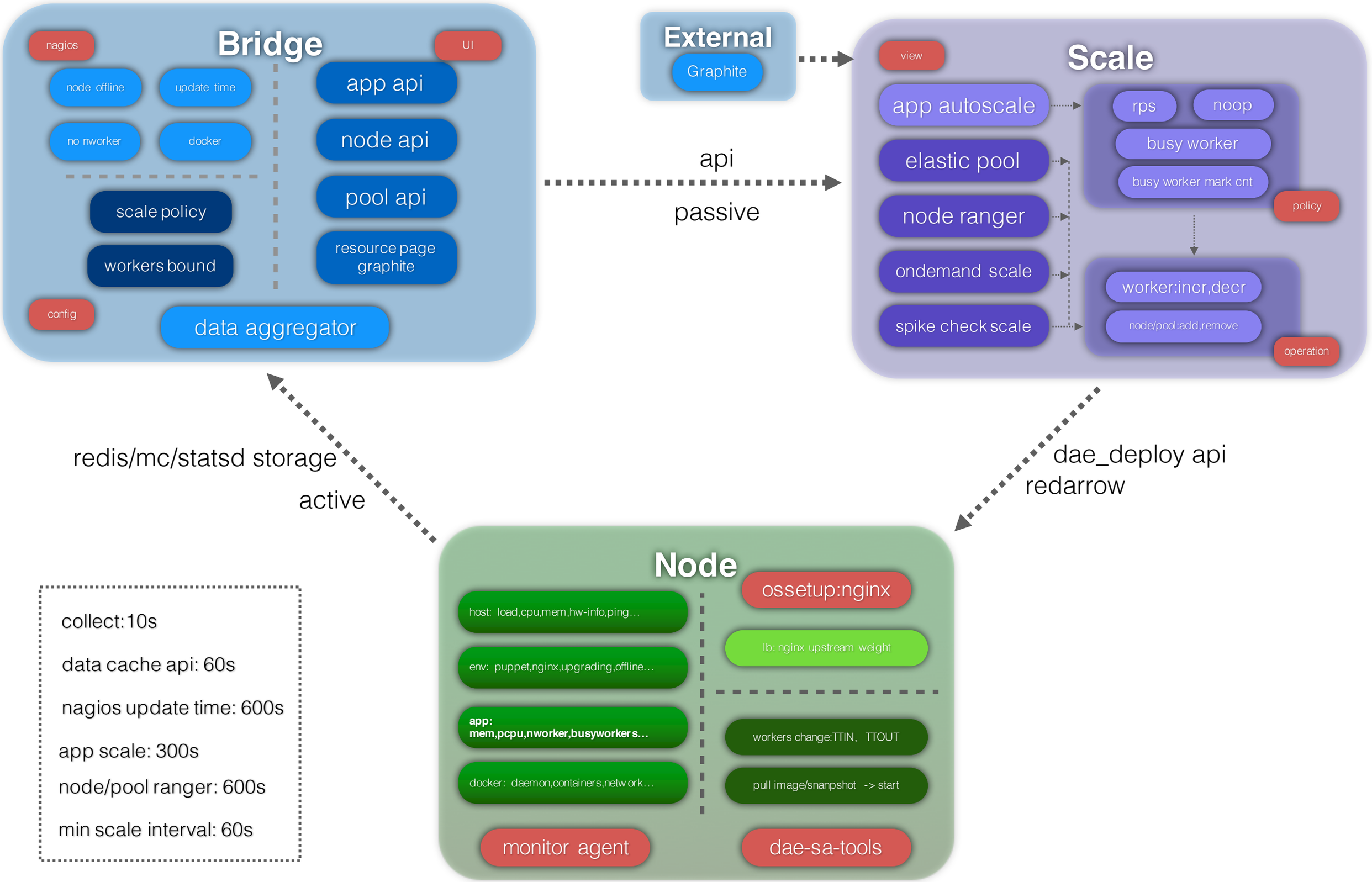
- **Gunicorn:**
  - web + thrift + pidl 的Python Server
  - Master + Worker 的多进程模型
  - worker 总数与应用服务处理能力直接相关

DAE scale 模型

- **Node:**
  - 物理节点，资源异构
  - 某些节点只部署单一app，但某些节点可能会部署160+个app
  - 从DAE角度Node只是MEM + CPU的资源单元，不会绑定或特别依赖某个具体的节点
  - Node与App是多对多关系，数据存储在bridge的数据库中

DAE scale 模型

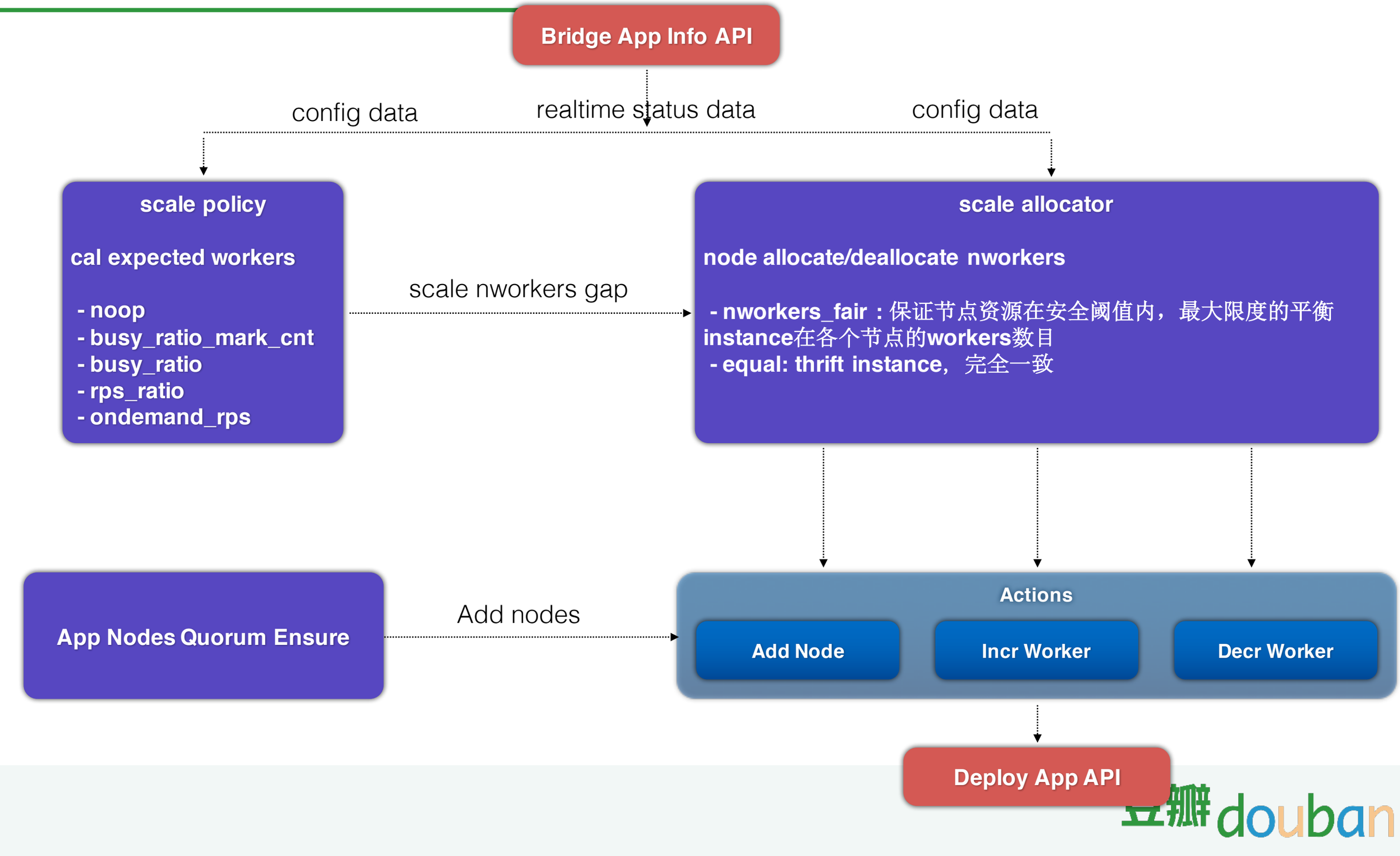
- **Pool:**
  - pool是一组节点的集合，pool与节点是1对多关系，实现node的分组管理与隔离
  - pool与应用是多对多关系
  - app诞生于default pool，会被 SA 手工迁移到其他pool中
  - 几个典型的pool：
    - default: app的默认pool，基本都是内网应用
    - production: 重要程度一般的外网应用
    - stable: 比较重要的外网应用和dae核心支撑应用
    - movie/frodo/group/sns: 每个pool中只有一个app
    - noscale/sa/dev: 功能pool，节点少
    - free: 为其他pool 在扩容时提供节点，缩容时容纳节点，没有任何app (但会跑mesos task)


Bridge
nagios
UI
node offline
update time
no nworker
docker
scale policy
workers bound
config
app api
node api
pool api
resource page
graphite
data aggregator
External
Graphite
api
passive
Scale
view
app autoscale
elastic pool
node ranger
ondemand scale
spike check scale
rps
noop
busy worker
busy worker mark cnt
policy
worker:incr,decr
node/pool:add,remove
operation
redis/mc/statsd storage
active
dae_deploy api
redarrow
Node
host: load,cpu,mem,hw-info,ping…
env: puppet,nginx,upgrading,offline…
app:
mem,pcpu,nworker,busyworkers…
docker: daemon,containers,network…
ossetup:nginx
lb: nginx upstream weight
workers change:TTIN, TTOUT
pull image/snanpshot -> start
monitor agent
dae-sa-tools
collect:10s
data cache api: 60s
nagios update time: 600s
app scale: 300s
node/pool ranger: 600s
min scale interval: 60s

采集数据

- 通过gunicorn worker是否busy来衡量当前负载能力
- busyworker mark cnt:
  - pre_request: touch worker mark file
  - post_request: rm worker mark file
  - mark file: /dev/shm/dae-busyworker-info/{appname}/{instance}/{pid}
  - monitor 在每台机器上定期检查和计数，并进行上报

Scale 应用策略


Bridge App Info API
config data
realtime status data
config data
scale policy
cal expected workers
- noop
- busy_ratio_mark_cnt
- busy_ratio
- rps_ratio
- ondemand_rps
scale nworkers gap
scale allocator
node allocate/deallocate nworkers
- nworkers_fair：保证节点资源在安全阈值内，最大限度的平衡instance在各个节点的workers数目
- equal: thrift instance，完全一致
App Nodes Quorum Ensure
Add nodes
Actions
Add Node
Incr Worker
Decr Worker
Deploy App API

Scale 应用策略

- **核心诉求**：调整单个app-instance的workers总数，并按照某种规则下发具体节点上（各个节点上instance workers数并不相同）
- **实现细节**：
  - vip apps 与 普通 apps 分离，用ThreadPool做并发执行
  - 若app处于stage0上线，需要额外进行处理
  - app scale是5min 一周期执行，但如果scale cron mq异常，导致cron任务积压，会通过与末次执行时间做检测，避免短期多次执行，引发scale 颠簸
  - 不同类型的instance有着不同的requirement_policy和allocator机制

Scale 应用策略

- 一些基本的策略：
  - 增加worker的时候，要快
  - 减少worker的时候，要缓
  - 增加节点的时候，要保证基本的处理能力
  - 避免应用在节点间分配过于不均匀
    - 引入app instance nworkers stdev(标准差)指标来做评测

Scale Pool 策略

- **目标:** 单个pool中各个节点资源均衡，避免某些节点应用过多、负载过大，造成机器宕机，让局部downtime减少
- **Balance Alg** :
  - classify nodes: cpu/mem/load, 按照low/normal/high 三个维度分类排序
  - ranger 权重顺序：cpu > mem > load
  - balance： busy <--> idle node 做balance， 将cpu/mem/load的差值折算成workers数量(物理节点上gunicorn workers资源总量/nworkers= resource per nworker)，再根据带调整nworkers数量规划出每个app移动多少nworkers。
  - 避免一个app大量占据该pool中的大部分节点，优先迁移整体workers数少的app，让大量workers的app留下。
  - balance时保证app nodes quorum规则不被破坏

Scale ElasticPool 策略

- **目标:** 保证pool总体负载合理
- **配置:** cpu/load/mem, low-high ratio
- **Pool:**
  - run: movie, group, sns, frodo (free pool 提供backup nodes支撑)
  - dryrun: production, stable, default
- **resize alg:**
  - > high: 一次性增加若干节点(1 <= x <= 5)
  - < low: 一次性只删减一个节点

Scale 策略流程 - Run Action部分

- **Deploy/Bridge 提供Authorized HTTP API**
  - http://dae_deploy.dapps.douban.com/api/{appname}/incr_worker
  - http://dae_deploy.dapps.douban.com/api/{appname}/decr_worker
  - http://dae_deploy.dapps.douban.com/api/{appname}/set_worker
  - http://dae_deploy.dapps.douban.com/api/{appname}/add_node
  - http://dae_deploy.dapps.douban.com/api/{appname}/remove_node
  - http://bridge.dapps.douban.com/node/{node_name}/change_pool

- 所有scale策略最后在执行时，都可以转化为对Action API的组合调用
- worker的调整通过redarrow 到具体物理节点上进行， 所以不可避免的会遇到执行失败的问题

监控

- **cron:**
  - ranger pool: (10min一次), 每个pool是独立的cron
  - resize pool: (5分钟一次)
  - autoscale app: (5分钟一次) (vip, ordinary 两个独立cron)
- **monitor**:
  - 所有scale相关cron监控max jobs和限制handler run timeout(一定的自动恢复)
  - graphite apps data: 过去10min中app instance的avg rps数据检查
  - eval_app: 模拟去掉app的负载最重的节点，看剩余worker是否能够承受过期15分钟的busy worker 30%的增长带来的负载
  - check scale crons running: 检查上次执行相关auto scale的时间戳，提前预警scale block问题

scale工具

- 迁移节点 / 增减worker的脚本
- 暂停scale的工具
- 详细的scale log便于查找问题

效果示例

豆瓣 douban

推而广之

离线的mq consumer是否也能auto scale？

Thx Q&A

Golang