

基于微服务的Real DevOps实践

锤子两年前登陆墨尔本时，完全是DevOps小白。面试REA的时候，被问到什么是Continuous Delivery（持续交付），锤子诚恳地表示“不知道”。面试官不依不饶，“不知道不要紧，你想想怎么样才能做到Continuous Delivery？”锤子回顾了一下自己维护项目时的噩梦，斟酌着用词说：“这需要好多自动化的工具支持，怎么测试，怎么接入不同的网络，怎么部署不同的环境，还有数据库的data migration和回滚，个顶个的都是痛点。”

之前锤子所做过的系统都是Monolith（单体）架构，所以，每次需要重启生产环境时，真得一而再，再而三地确认。所以当REA的同事给我讲解我们的微服务架构，和怎么做Continuous Delivery时，我问：“那最坏的情况就是重启系统了？”这个同事说：“不，重启系统是正常情况。”由此，锤子开始慢慢了解REA强大的DevOps。

微服务和DevOps

在介绍REA的DevOps之前，还是简单说说我们使用的微服务架构。

微服务有时被人诟病违背了DRY原则，但是Monolith架构下各种服务间的强耦合对于扩展部署都很痛苦，所以Don't repeat yourself 诚然不错，微服务架构下，定义好边界之后（APIs），每个微服务独立存在，独立部署且可扩展，即使有一些简单的复制粘贴，其带来的灵活性也是Monolith架构不具备的。

在REA内部，为了保证某种程度的一致性，我们的微服务都需要遵循12 Factor：

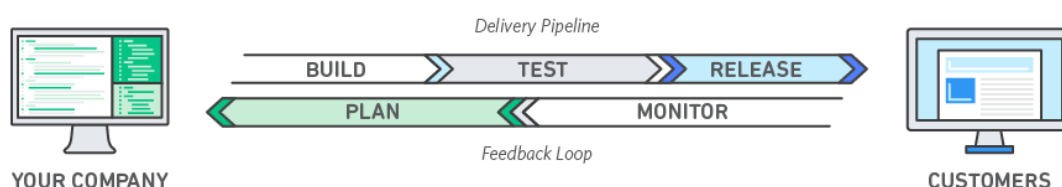
- 基准代码：一份基准代码，多份部署。
- 依赖：显式声明依赖关系。
- 配置：在环境中存储配置。
- 后端服务：把后端服务当作附加资源。
- 构建，发布，运行：严格分离构建和运行。
- 进程：以一个或多个无状态进程运行应用。
- 端口绑定：通过端口绑定提供服务。
- 并发：通过进程模型进行扩展。
- 易处理：快速启动和优雅终止可最大化健壮性。

- 开发环境与线上环境等价：尽可能的保持开发，预发布，线上环境相同。
- 日志：把日志当作事件流。
- 管理进程：后台管理任务当作一次性进程运行。

不过不管微服务设计得如何精良，当一个“小而美”的团队（5-6名开发人员）需要同时开发维护生产环境中10个以上的微服务时，服务运行和管理上的额外复杂性使得全自动构建和部署变得不可或缺，更进一步，最好使用Continuous Delivery来解决这些问题，而DevOps为真正的Continuous Delivery提供了必要的支持。

那DevOps究竟是什么？DevOps就是更好的优化开发(DEV)、测试(QA)、运维(OPS)的流程，开发运维一体化，通过高度自动化工具与流程来使得软件构建、测试、发布更加快捷、频繁和可靠。

下面这张图（图片来源[什么是 DevOps ?](#)）就是DevOps的日常。



既然DevOps不是新工具，不是新团队，不是新角色，也不是大量知识，而且DevOps的伟大实践来自于许多不同的领域，并且在IT组织内执行，为了提升IT的性能，那下面就让锤子讲讲REA的DevOps实践，首先自然从最贴近我们程序猿的全工具链开始。

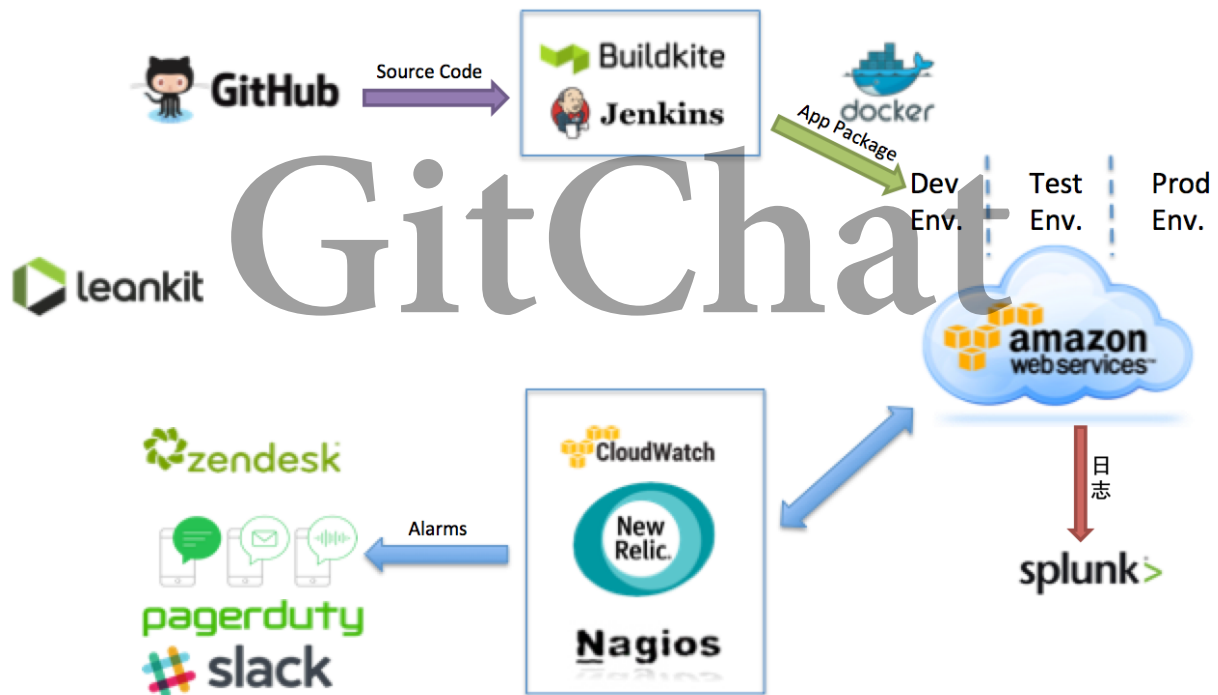
REA DevOps全工具链

REA实际使用到的全工具链包括：

- 代码仓库：github企业版。
- 构建和部署工具：Buildkite和Jenkins，还有一些老的项目依然在使用Bamboo。
- 容器平台：微服务用Docker进行打包，Docker的引入让公司内部微服务的部署流程一致化。
- 环境：目前绝大部分的service部署在AWS（Amazon Web Services）中，开发环境和测试环境使用同一套IAM，生产环境使用另一套IAM，三套环境通过Virtual Private Cloud (VPC)的设置进行隔离。不同部门的不同的组在IAM下拥有不同角色和访问权限。部署采用AWS Cloudformation服务，避免手动创建和更新使用到的AWS的业务。
- 日志管理：splunk。Docker已经提供对Splunk的支持，所有微服务的日志都能够通过Splunk Agent发送到集中的Splunk服务器。集中式的日志管理方式，方便程序猿

们进行trouble shooting，而且完全避免了登陆到不同的机器收集log的窘境。

- 监控：建立分层的监控体系。使用NewRelic用来监控网络、设备和应用的性能；使用AWS Cloudwatch监控AWS的服务，比如某个SQS对应的dead-letter queue里是不是收到了消息，或者AWS Lambda执行是不是有错误等；Nagios提供服务可用性监控。可以直接使用REA内部的rea-health-check库，提供心跳API供Nagios的主动模式使用；也可以通过设置接收相应的消息格式，使用Nagios被动模式监控微服务。不管哪种模式下发现问题，Nagios马上会产生告警发送到PagerDuty。
- 告警：PagerDuty。前面提到的监控工具都可以发出告警，这些告警会通过PagerDuty用邮件，Slack，电话和短信的方式通知给当时的值日人员。PageDuty虽然是全天24小时运行，也只有极少部分高优先级的告警才会在非工作时间发出。还有一个告警的来源是Zendesk Ticket，这种信息通常直接来自于客户。就我们组而言，基本上都是一些数据错误，需要个别修正。
- 协同工作：Leankit。



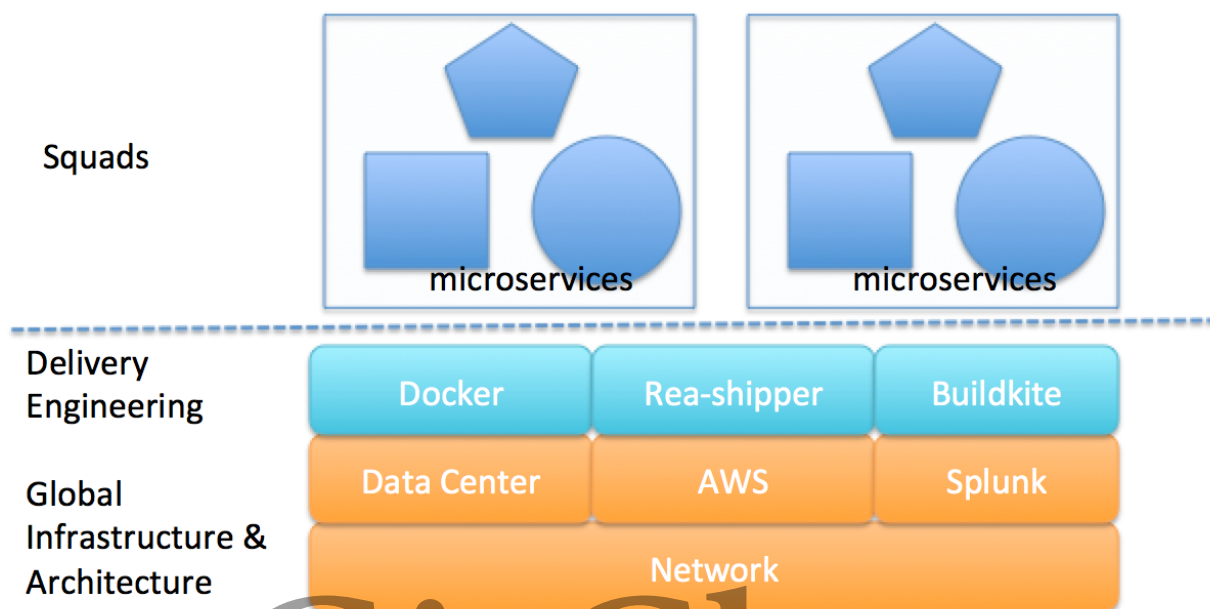
工具链看起来并不复杂，那么有了工具链，所有的问题就迎刃而解了吗？答案自然是否定的。

怎么样自动化所有流程？怎样为REA超过40个组管理AWS的账号？怎么为AWS里的测试环境和生产环境配置VPC？怎样把微服务部署到不同的环境？监控的策略怎么样实施，怎么样在不同的组之间协调？所有的几百个微服务，一旦某些微服务除了问题怎么办等等。这些都需要工具链切切实实的落地。

REA DevOps工具链的实施过程中，分层协作不但厘清不同部门的责任，也让所有开发人员参与到Operation工作中，让DevOps融入每个IT人员的日常。

REA DevOps的分层协作

REA DevOps不同层面的问题由不同的组负责，如下图。

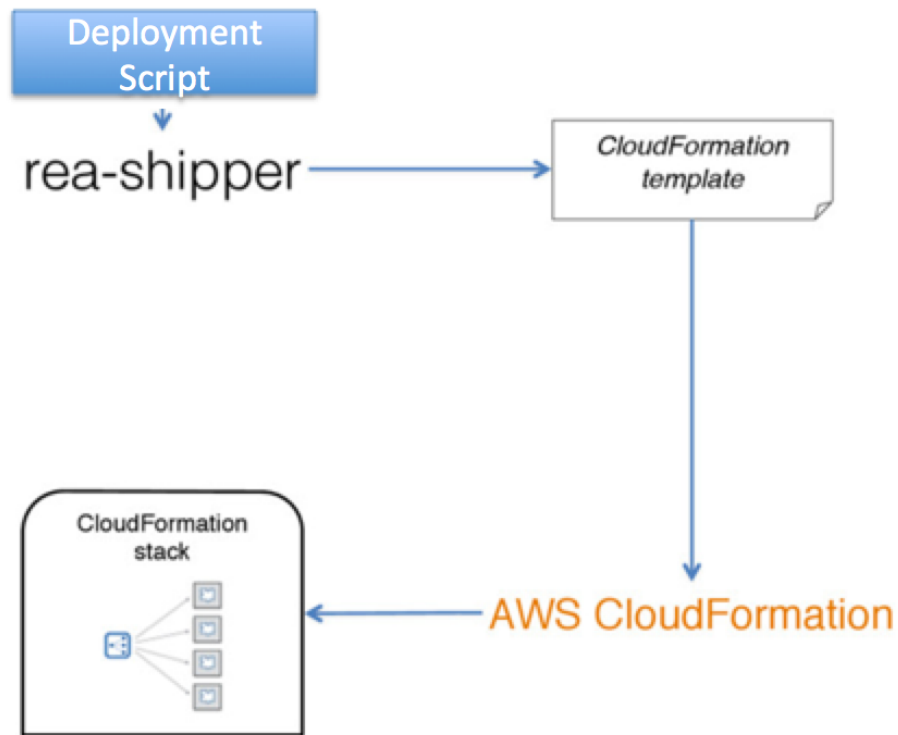


不同的Squads（ REA引入了 [Spotify模式](#) ）使用DevOps工具链来开发和维护自己的微服务。详情会在下面一节进行介绍。

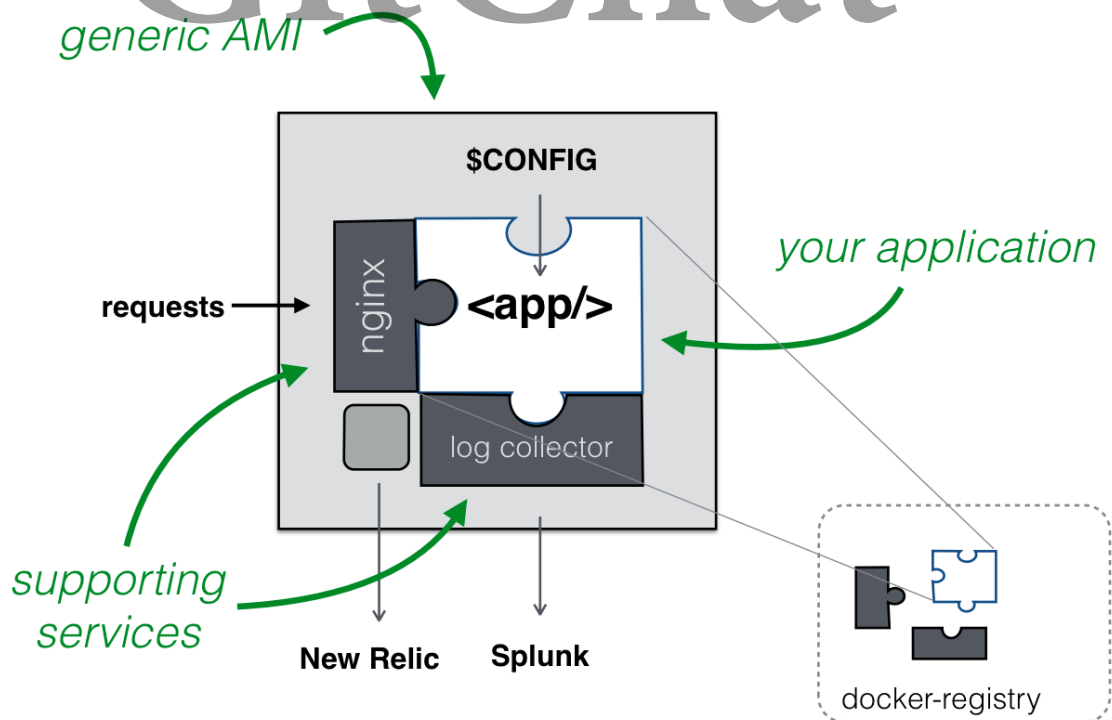
Delivery Engineering团队，顾名思义，就是为了让程序猿们更快更好地发布应用，主要职责是工具的开发和管理。包括创建Docker Registry，准备好已经安装了必要库的**baked Docker image**，有了这些基准的Docker Images，开发团队在为自己的微服务构建Docker Image的时可以有效节省时间，还能规避各种库版本不一致的问题。Delivery Engineering还需要为不同部门配置Buildkite，Buildkite的Agents部署在AWS的EC2中，根据需要连接的环境，多个Agents分布在不同的VPC。

最值得一提的是Delivery Engineering开发的[rea-shipper](#)。

- rea-shipper实现了从构建到部署的全自动，真正解决了最痛的那个点。
- 为了保证Zero downtime，rea-shipper读取微服务的cloudformation配置，采用**immutable ASG (auto scaling group) deployment模式**进行部署。



- re-a-shipper自动将监控和日志管理所需要的模块也用Docker打包，在部署过程中与每个微服务的Docker Image整合，如下图所示。这样微服务开发人员只需要关注业务的开发，无需担心日志和监控的问题。



以我们一个实际的service charge-central为例，登陆到EC2 instance之后，可以看到正在运行的几个container。

```

ubuntu@i-03:~$ docker ps
CONTAINER ID        IMAGE
e17ad69b2cbe       r[REDACTED].realstate.com.au/cowbell/nginx-reverse-proxy:201702031327
03da7a472aae       r[REDACTED].realstate.com.au/cowbell/ec2-instance-metrics-collector:201702141012
29b9910eccec       7[REDACTED].dkr.ecr.us-east-1.amazonaws.com/[REDACTED]/charge-central:1.0.282
5588d0f3ab2e       r[REDACTED].realstate.com.au/cowbell/rea-shipper-log-router:201610211041
fc56e4ae7222       r[REDACTED].realstate.com.au/cowbell/splunk-universal-forwarder:6.5.0
ubuntu@i-03:~$

```

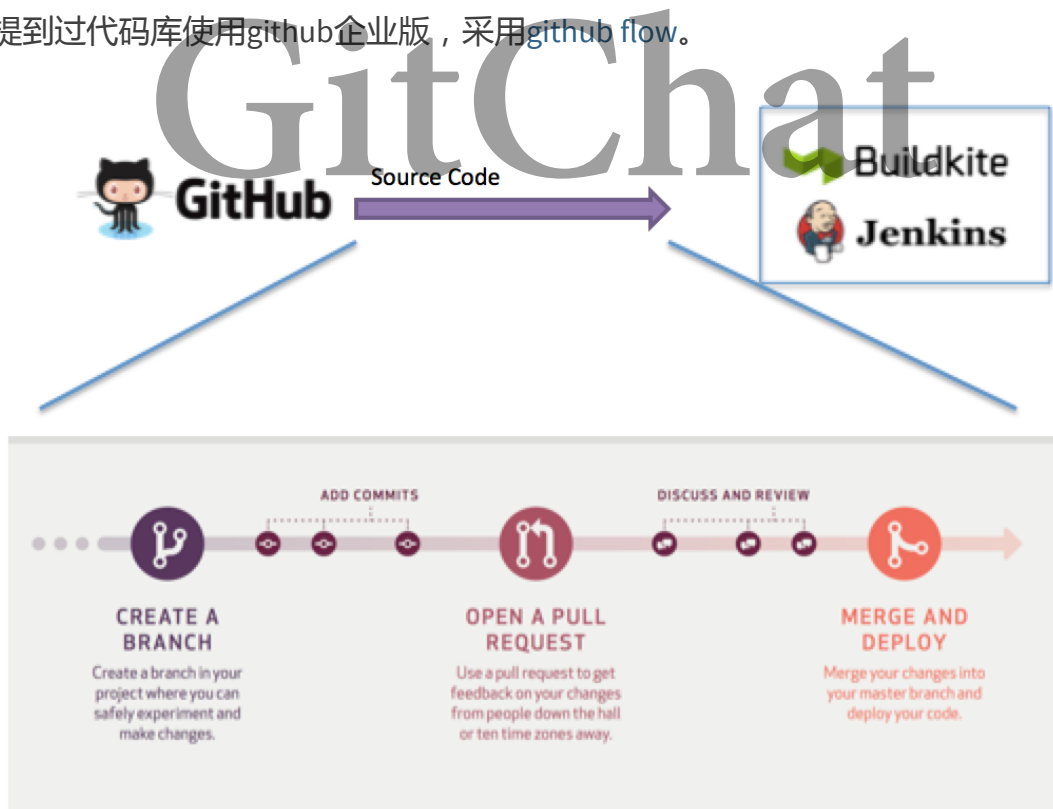
Global Infrastructure & Architecture 团队负责基础设施建设，管理维护并开发相应的工具，包括网络、数据中心、AWS账号的管理、Splunk的集成等。比如，去年悉尼大雨，Amazon的机房被水淹了，澳洲大批网站受到影响，也包括我们公司的一小部分，当时苦了GIA team的人了。

所以全工具链的配置和开发是Global Infrastructure & Architecture 和Delivery Engineering 的职责，而作为程序猿的锤子，则是工具链的使用者。下面就说说程序猿所参与的DevOps日常。

程序猿DevOps日常

1. Coding

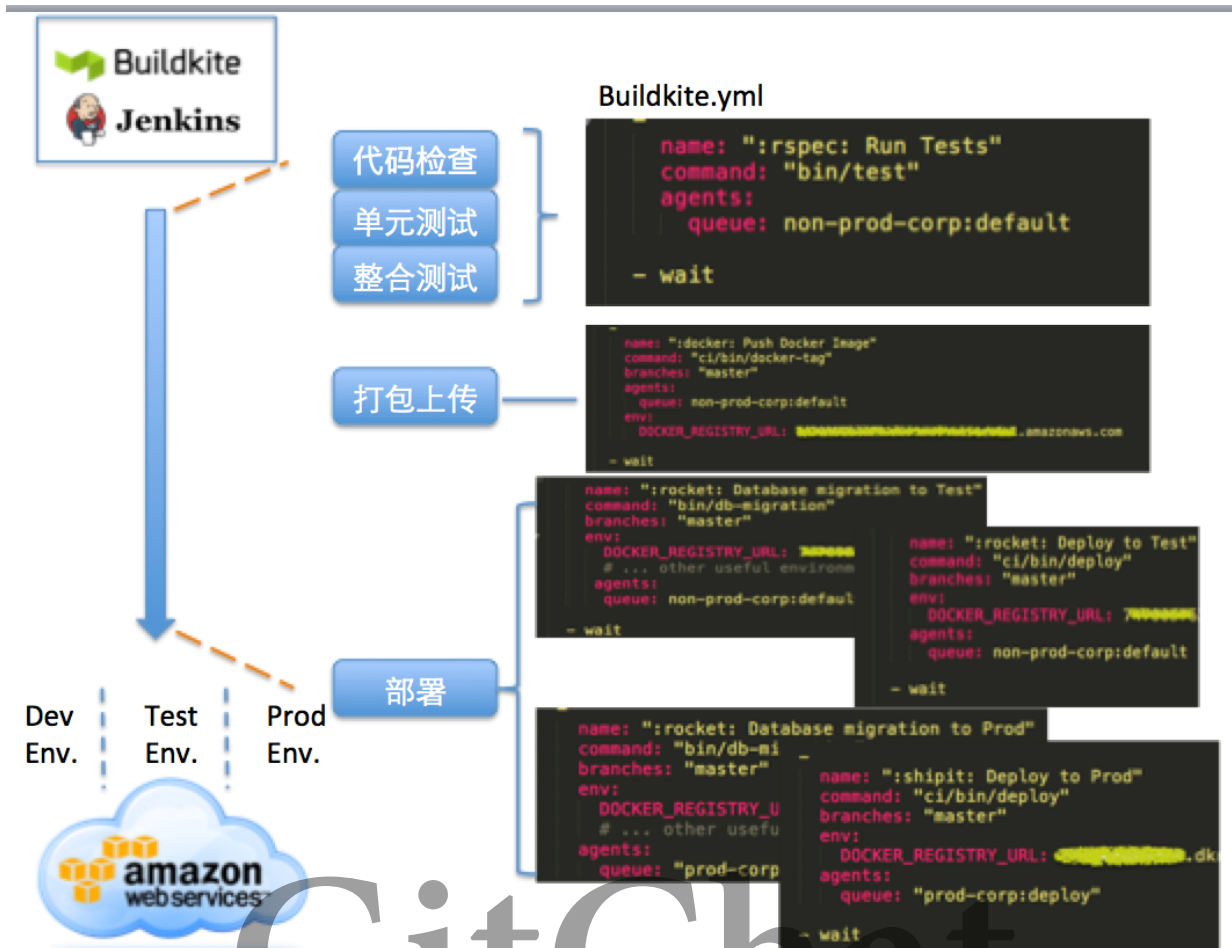
前面提到过代码库使用github企业版，采用github flow。



微服务需要的部署脚本和AWS Cloudformation的信息，提供给不同监控工具的接口，[fried Docker image](#) 定义等，也都是代码的一部分，需要开发人员完成。

2. 构建和部署

构建和部署就以Buildkite为例子，这是一个微服务的buildkite脚本。



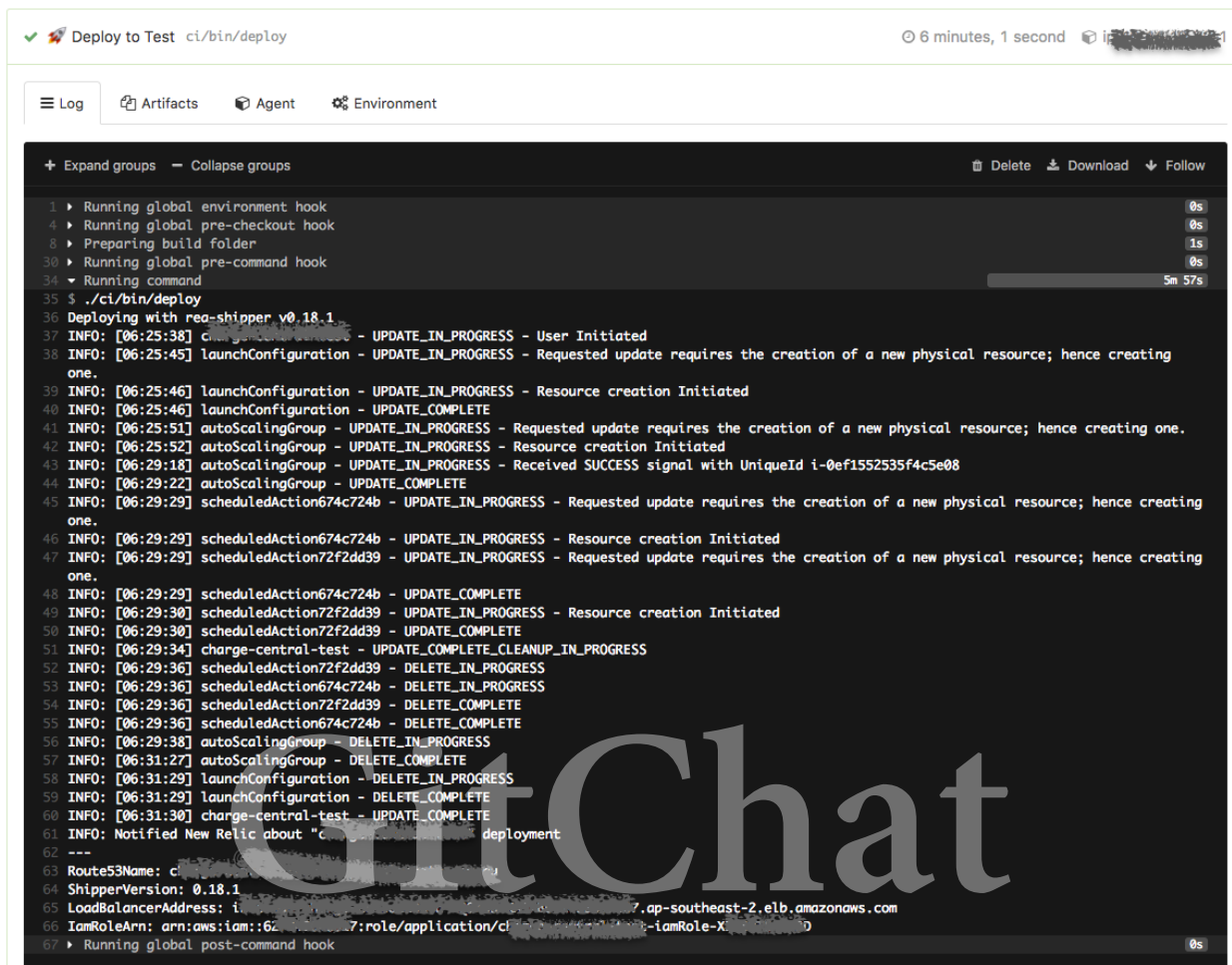
这个流程里代码检查和单元测试自动化起来很容易，那么怎么做整合测试？REA基于Ian Robinson提出的[用消费者驱动的契约进行面向服务开发的模式](#)开发了 开源的[Pact 测试框架](#)，用轻量级的契约测试来代替厚重的集成测试。Pact在消费端用单元测试的形式（更轻）来生成 pact 契约，服务端通过验证契约来保证两者稳定集成。一旦有一端契约未经协商发生改变，那么Pact测试就会失败。

构建成功之后，会把微服务打包成Docker Image然后上传到Docker Registry。我们会选择在Delivery Engineering提供的基准Docker Image之上来打包，这是一个微服务的Dockerfile的例子：

```
FROM registry.ubuntu.com:12. realstate.com.au/ci/12.1/ubunt-ruby2.2:201512181702
RUN apt-get update -y
RUN apt-get install -y libpq-dev postgresql-client
WORKDIR /app
ADD Gemfile /app/
ADD Gemfile.lock /app/
RUN bundle install --jobs 8 --retry 3
ADD . /app/
EXPOSE 80
CMD bin/run
```

用这种方式，在buildkite上打包并上传到Docker Registry的时间小于三分钟。

部署时，脚本会调用rea-shipper。不同的环境下，Buildkite会选择不同的Agent进行部署：Test环境的non-prod-corp:default和Prod环境的prod-corp:default。部署的时间通常10分钟以内，下面是一个微服务部署到test（6分1秒）和prod（5分43秒）的时间，图中能够看到Cloudformation更新的步骤。



The screenshot shows a Buildkite deployment interface for a job named 'Deploy to Test' using the 'ci/bin/deploy' script. The deployment is completed in 6 minutes and 1 second. The log output shows the following steps:

- 1 ▶ Running global environment hook (0s)
- 4 ▶ Running global pre-checkout hook (0s)
- 8 ▶ Preparing build folder (1s)
- 30 ▶ Running global pre-command hook (0s)
- 34 ▶ Running command (5m 57s)
- 35 \$./ci/bin/deploy
- 36 Deploying with rea-shipper v0.18.1
- 37 INFO: [06:25:38] charge-central-test - UPDATE_IN_PROGRESS - User Initiated
- 38 INFO: [06:25:45] launchConfiguration - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
- 39 INFO: [06:25:46] launchConfiguration - UPDATE_IN_PROGRESS - Resource creation Initiated
- 40 INFO: [06:25:46] launchConfiguration - UPDATE_COMPLETE
- 41 INFO: [06:25:51] autoScalingGroup - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
- 42 INFO: [06:25:52] autoScalingGroup - UPDATE_IN_PROGRESS - Resource creation Initiated
- 43 INFO: [06:29:18] autoScalingGroup - UPDATE_IN_PROGRESS - Received SUCCESS signal with UniqueId i-0ef1552535f4c5e08
- 44 INFO: [06:29:22] autoScalingGroup - UPDATE_COMPLETE
- 45 INFO: [06:29:22] scheduledAction674c724b - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
- 46 INFO: [06:29:29] scheduledAction674c724b - UPDATE_IN_PROGRESS - Resource creation Initiated
- 47 INFO: [06:29:29] scheduledAction72f2dd39 - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
- 48 INFO: [06:29:29] scheduledAction674c724b - UPDATE_COMPLETE
- 49 INFO: [06:29:30] scheduledAction72f2dd39 - UPDATE_IN_PROGRESS - Resource creation Initiated
- 50 INFO: [06:29:30] scheduledAction72f2dd39 - UPDATE_COMPLETE
- 51 INFO: [06:29:34] charge-central-test - UPDATE_COMPLETE_CLEANUP_IN_PROGRESS
- 52 INFO: [06:29:36] scheduledAction72f2dd39 - DELETE_IN_PROGRESS
- 53 INFO: [06:29:36] scheduledAction674c724b - DELETE_IN_PROGRESS
- 54 INFO: [06:29:36] scheduledAction72f2dd39 - DELETE_COMPLETE
- 55 INFO: [06:29:36] scheduledAction674c724b - DELETE_COMPLETE
- 56 INFO: [06:29:38] autoScalingGroup - DELETE_IN_PROGRESS
- 57 INFO: [06:31:27] autoScalingGroup - DELETE_COMPLETE
- 58 INFO: [06:31:29] launchConfiguration - DELETE_IN_PROGRESS
- 59 INFO: [06:31:29] launchConfiguration - DELETE_COMPLETE
- 60 INFO: [06:31:30] charge-central-test - UPDATE_COMPLETE
- 61 INFO: Notified New Relic about "charge-central-test" deployment
- 62 ---
- 63 Route53Name: ci
- 64 ShipperVersion: 0.18.1
- 65 LoadBalancerAddress: i-0ef1552535f4c5e08.ap-southeast-2.elb.amazonaws.com
- 66 IamRoleArn: arn:aws:iam::627464227777:role/application/ci
- 67 ▶ Running global post-command hook (0s)

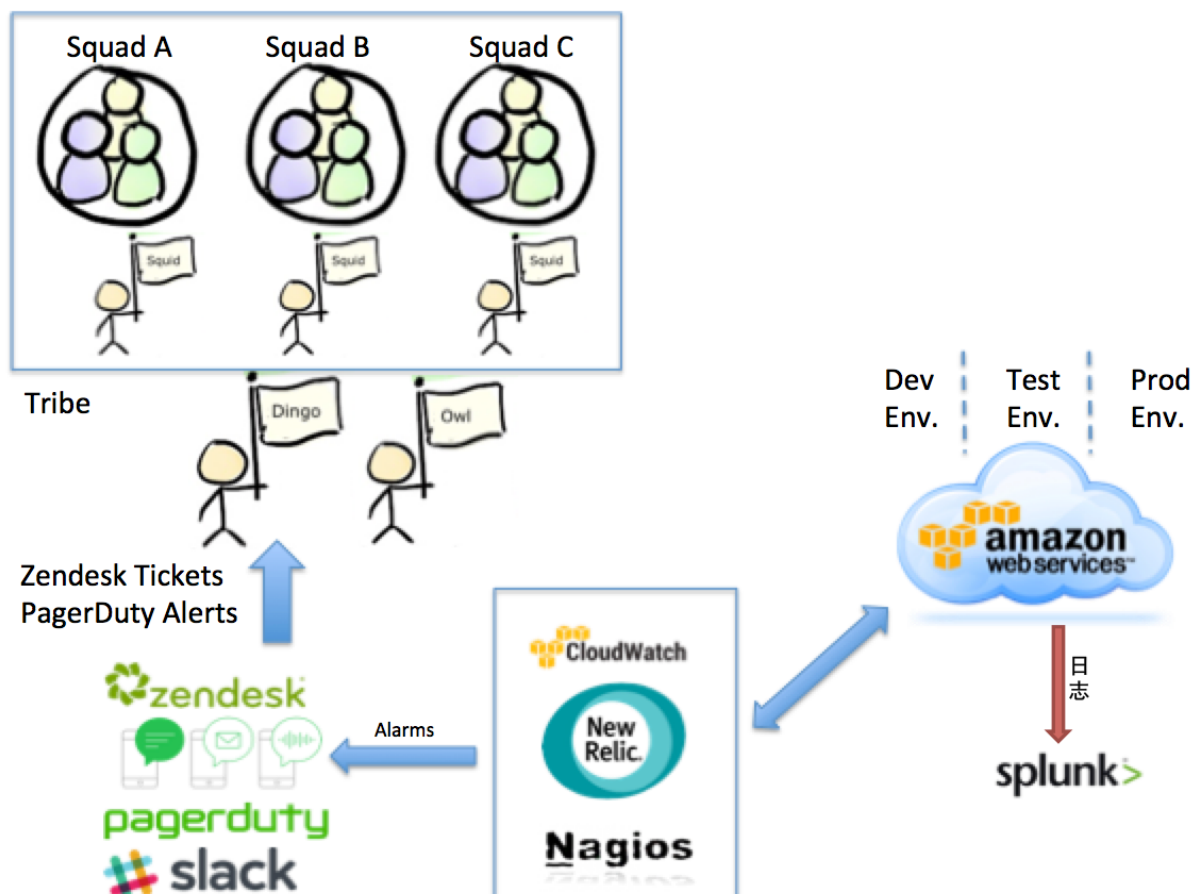
The screenshot shows a deployment log for a production environment. The log is titled 'Deploy to Prod' and shows a successful deployment of 'rea-shipper v0.18.1'. The log includes various status messages such as 'UPDATE_IN_PROGRESS', 'UPDATE_COMPLETE', 'DELETE_IN_PROGRESS', and 'DELETE_COMPLETE' for different resources like 'launchConfiguration', 'autoScalingGroup', and 'scheduledAction'. The log also shows the 'Route53Name' and 'ShipperVersion'.

```
✓ Deploy to Prod ci/bin/deploy 5 minutes, 43 seconds
Log Artifacts Agent Environment
+ Expand groups - Collapse groups Delete Download Follow
1 ▶ Running global environment hook 0s
4 ▶ Running global pre-checkout hook 0s
8 ▶ Preparing build folder 1s
30 ▶ Running global pre-command hook 0s
34 ▶ Running command 5m 40s
35 $ ./ci/bin/deploy
36 Deploying with rea-shipper v0.18.1
37 INFO: [06:32:47] ch... - UPDATE_IN_PROGRESS - User Initiated
38 INFO: [06:32:54] launchConfiguration - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
39 INFO: [06:32:55] launchConfiguration - UPDATE_IN_PROGRESS - Resource creation Initiated
40 INFO: [06:32:55] launchConfiguration - UPDATE_COMPLETE
41 INFO: [06:33:01] autoScalingGroup - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
42 INFO: [06:33:02] autoScalingGroup - UPDATE_IN_PROGRESS - Resource creation Initiated
43 INFO: [06:36:09] autoScalingGroup - UPDATE_IN_PROGRESS - Received SUCCESS signal with UniqueId i-0804f29e3a7a28b27
44 INFO: [06:36:11] autoScalingGroup - UPDATE_COMPLETE
45 INFO: [06:36:16] scheduledAction82dac073 - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
46 INFO: [06:36:16] scheduledAction82dac073 - UPDATE_IN_PROGRESS - Resource creation Initiated
47 INFO: [06:36:16] scheduledAction2a3c7d3d - UPDATE_IN_PROGRESS - Requested update requires the creation of a new physical resource; hence creating one.
48 INFO: [06:36:16] scheduledAction82dac073 - UPDATE_COMPLETE
49 INFO: [06:36:17] scheduledAction2a3c7d3d - UPDATE_IN_PROGRESS - Resource creation Initiated
50 INFO: [06:36:17] scheduledAction2a3c7d3d - UPDATE_COMPLETE
51 INFO: [06:36:24] charge-central - UPDATE_COMPLETE_CLEANUP_IN_PROGRESS
52 INFO: [06:36:27] scheduledAction2a3c7d3d - DELETE_IN_PROGRESS
53 INFO: [06:36:27] scheduledAction2a3c7d3d - DELETE_COMPLETE
54 INFO: [06:36:27] scheduledAction82dac073 - DELETE_IN_PROGRESS
55 INFO: [06:36:28] scheduledAction82dac073 - DELETE_COMPLETE
56 INFO: [06:36:29] autoScalingGroup - DELETE_IN_PROGRESS
57 INFO: [06:38:18] autoScalingGroup - DELETE_COMPLETE
58 INFO: [06:38:20] launchConfiguration - DELETE_IN_PROGRESS
59 INFO: [06:38:20] launchConfiguration - DELETE_COMPLETE
60 INFO: [06:38:21] charge-central - UPDATE_COMPLETE
61 INFO: Notified New Relic about "charge-central" deployment
62 ---
63 Route53Name: ch...
64 ShipperVersion: 0.18.1
65 LoadBalancerAddress: ...620.ap-southeast-2.elb.amazonaws.com
66 IamRoleArn: arn:aws:iam::74...9:role/application/c...1-iamRole-E8S7Z1I5WJVD
67 ▶ Running global post-command hook 0s
Back to top
```

尽管是全自动的部署，考虑到生产环境的重要性，我们还是选择谨慎地Block，需要某个开发人员手动触发。触发的时间没有特别的规定，只是在我们的kanban中，deploy是最后一步，这意味着只有真正部署到生产环境，这个卡片才算完成。如果部署过程中出现失败，rea-shipper不会切换运行中的ASG（Auto Scalling Group），业务并不会受到影响。如果部署的新版本发现bug需要紧急回滚，可以很容易地根据Docker Image的版本找到相应的Image进行部署。

3. 运维

日常的运维如下图所示：



需要处理的问题一般有两种：

- 直接源于客户的问题，使用Zendesk Ticket。
- 源于生产环境的问题，比如PagerDuty告警。

我们Tribe有5个Squad，除了有超过30个microservice之外，还有跟不同系统的接口，如果不能组织好，开发人员每天必定会被各种问题打扰。所以如图所示，Tribe级别有Dingo（工作时间）或者Owl（非工作时间）作为接口人，负责处理和分发问题到Squad级别的Squid。Dingo，Owl和Squid是团队的开发人员轮岗。

总结

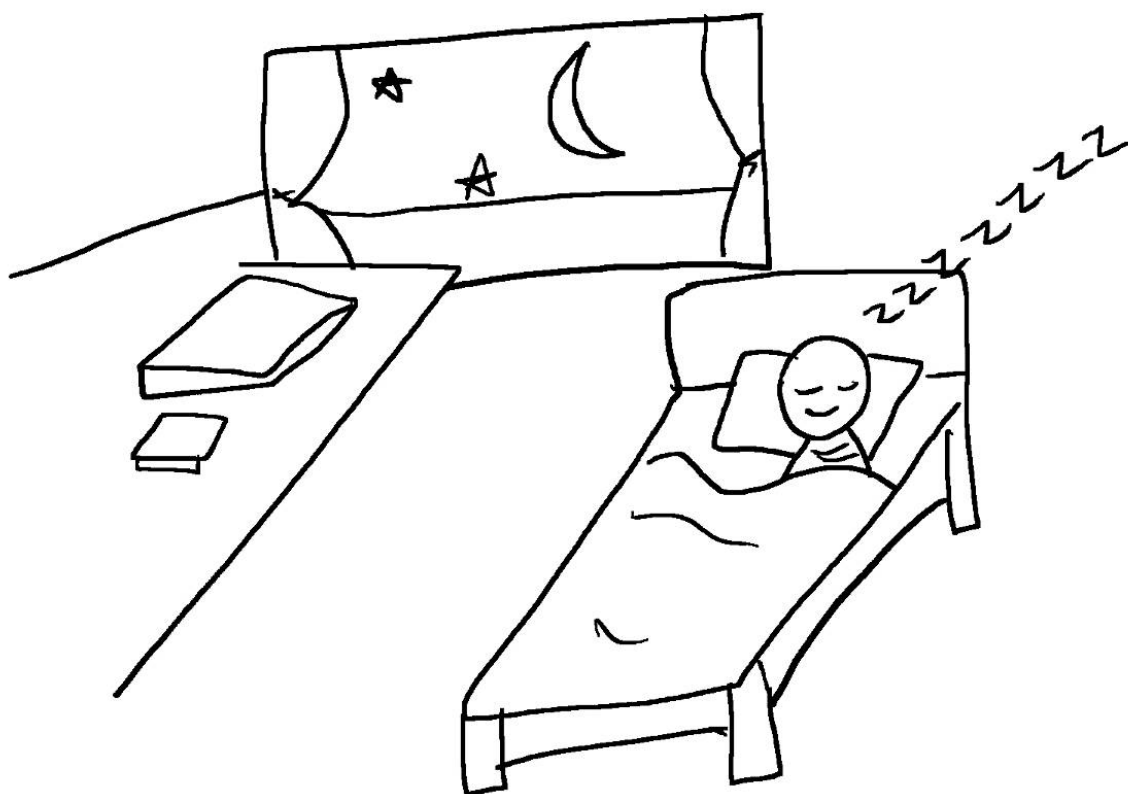
本文介绍了REA DevOps的实践，包括工具链，工具链的分层协作以及使用中的流程。再来对比一下Gene Kim的3个方法：流程，反馈和持续学习，这3个方法是DevOps的主要部分，提供一种路标来理解和执行DevOps。锤子能够看到的是在REA DevOps实践中，每个开发人员都参与到流程的不断优化中，让流程变得更顺畅和快速；通过不同方式可视化监控和反馈，以达到更快的反馈路径；开放全代码库给所有开发人员，鼓励程序猿持续学习和改进等等。

以上种种，推荐阅读我们公司同事的文章来更深入的了解REA的文化。Scaling On-Call: from 10 Ops to 100 Devs，讲述了怎么从这样的状态：



到达下面的状态：

GitChat



这种变化并不是技术改进带来的，而是源于持续学习的企业文化。而这，正是DevOps最需要的。

在此，特别感谢任发科对本文的再三评审和宝贵意见，感谢谢工和李盼的鼓励，感谢徐毅的推荐！

GitChat