解读 ChatOps：开源聊天机器人 Hubot 如何协助运维?

作者：顾伟
阅读数：6414 2016 年 11 月 3 日 话题：语言 & 开发 架构 运维

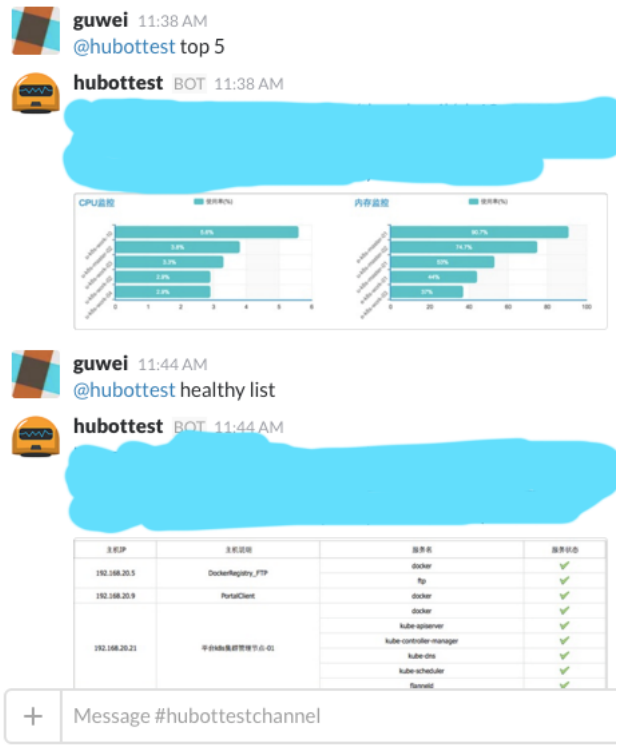
ChatOps 通常是指依靠群组聊天室进行管理运维工作的一种。在 ChatOps 领域，我是一个新人，通过学习与运用，再回过头来看，对 Github、Apple 这样的一些先行者更是崇拜。

在现在这个概念为王的时代，ChatOps 更像是一个"弱建筑"定义，低调不失优雅。希望通过我的分享，和大家一起来发现其生态建设（以我熟悉的 Hubot 为例）、基本设计，为后续更好的实践提供一个参考。

背景，何为 ChatOps?

先看看实验室截图，我在聊天室中通过与某机器人沟通，获取容器云的测试环境的 top5 资源以及主机健康信息表。

(点击放大图像)



直观的感受就是 ChatOps 给了一个全新的工作环境，让我们可以在聊天室中，通过聊天的方式，获取想要的反馈。

说到 ChatOps，自然会想到 DevOps。市场上这两年在"疯狂"的传递着 DevOps 的理念，那我们有没有考虑过 DevOps 的核心是什么？有哪些实现分支？又存在一些什么问题？很多人都像我一样，会习惯的去说，DevOps 有四大核心，包括技术、组织、流程、文化；实现 DevOps 可以从 CI/CD 着手，以自助自动为指导思想；DevOps 要落地很难，因为有太多历史债务，有太多规章制度.....

on CAMS”。这里，CAMS 是指 a culture of automation, measurement and sharing，认为 ChatOps 是对 DevOps 的一个实现与加强。

一直以来，运维的工作方式给大家的感觉就是脚本，部署要执行脚本、变更要执行脚本；或者进阶一层来看，运维会用各种小工具，比如 Puppet、SaltStack 等，对脚本形成统一管理、下发、执行。很多人都在讲：要把繁重且重复的劳动交给机器人，让人做更有趣更创新的事情。比如运维同学所做的日常巡检、故障处理，则可以由这些机器人伙伴来协助处理。而作为运维同学的伙伴机器人，一个很好的参与工作方式是加入到我们的日常聊天组，一起共事、一起学习。----- 这就是 ChatOps，但不局限于 Ops。

低调，互联网时代的另类

现在市场上 ChatOps 的开源实现，呈三足鼎立之势：

1. Hubot：CoffeeScript 实现，Github 提供且自用
2. Lita：Ruby 实现，支持容器部署，依赖 redis
3. Err：Python 实现，笔者目前还没有用过

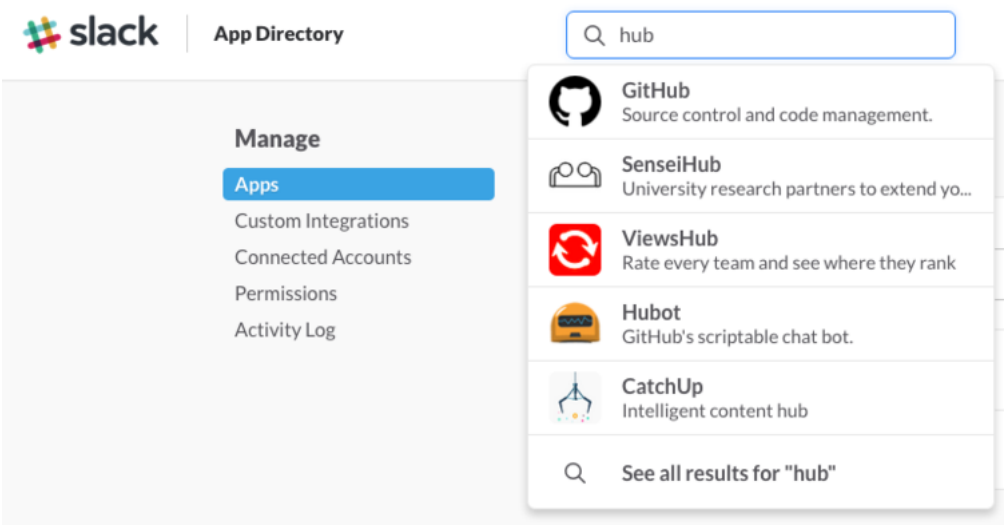
以 Hubot 为例，这是 Github 在 5 年多前开发的一套用于管理 Github 自己的软硬件的机器人，中间历经了自用、开源、重写再开源三个阶段，现在俨然成为 Github 上最火热的项目之一：

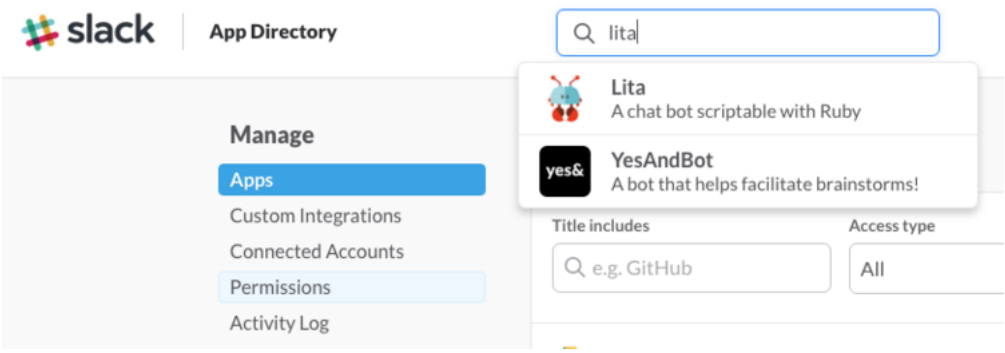
在过去的 5 年多时间中，其宣传相比 DevOps、微服务、容器这些概念来说，少的可怜；但是最近 ChatOps 更像是被推出到了风口，被越来越多的提及到了。

开源生态讲究的是横纵连横，单向集成是生态建设的最大阻力。在第一次使用 Hubot 时，其生态建设的完备性相当让我出乎意料，在出向上，Hubot 本身已适配很多：

- Shell, i.e. for use with develop
- Campfire
- Visual Studio Online
- Weixin
- XMPP
- Yammer
- ChatWork
- Dasher
- Fleep
- Flowdock
- Gitter
- Gtalk
- Hall
- HipChat
- iMessage
- IRC
- Jabbr
- Let's Chat
- Lingr
- Mattermost
- Partychat
- Proxy - This adapter a defined in a config ob
- Rocket.Chat
- Slack
- Skype
- SkypeWeb
- Skyweb
- Talker
- Telegram
- Twilio IP Messaging
- Twilio SMS
- Twitter
- Typetalk
- VictorOps

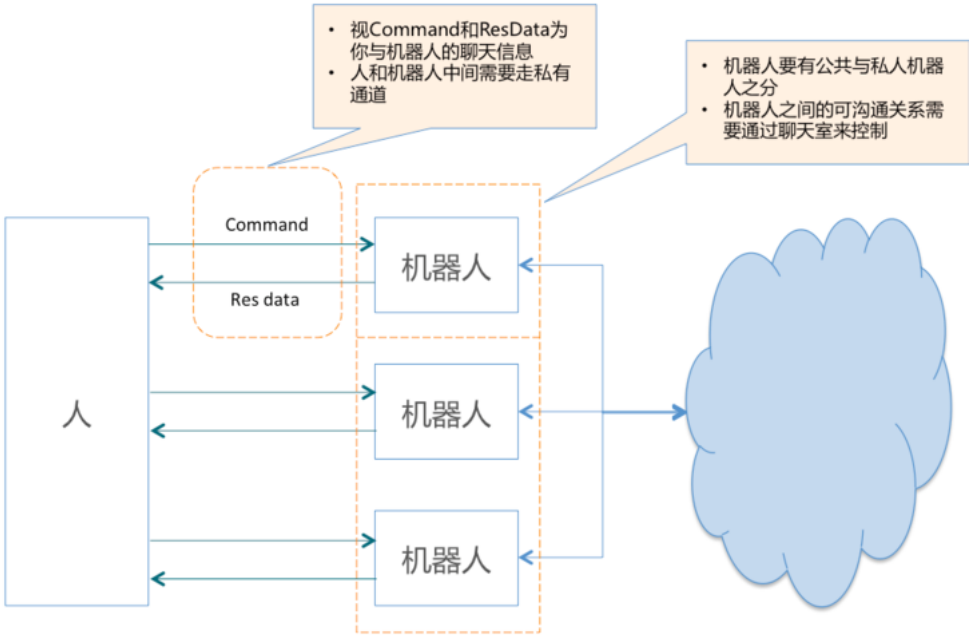
而在入向上，我使用的 Slack、HipChat 都默认地做了对 Hubot 的集成。以 Slack 为例，进入应用管理后，直接就可以集成 Hubot、Lita，而不需要自己通过 API 做集成了。





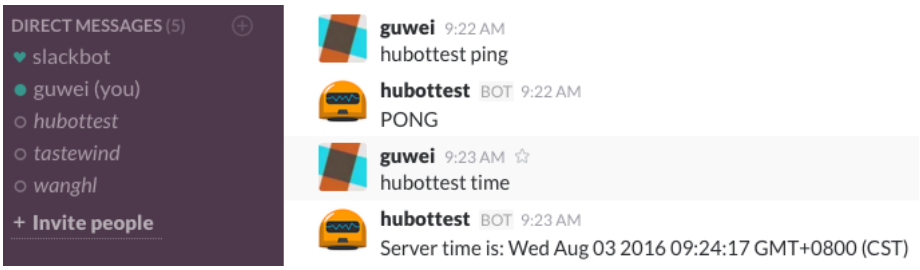
除了上面提到的与 chat 软件的集成，在部署环境上，Unix、Windows 都可支持，而且 Hubot 支持了 Azure、Bluemix、Heroku 等云环境的快速部署（虽然还没全自动化）。这让我不由地想到：国内的一朵朵公有云，天天在谈生态，为什么不考虑去做这些事情呢……

机器人，说的少做的很多



- 通过给予 command，由机器人伙伴去实际云中操作，人和机器人伙伴的通信走私有通道，机器人伙伴会将信息回复到聊天室中。
- 机器人伙伴同样分公共与私人的，最简单的方式就是用不同聊天室来隔离（不同的圈子嘛）。

机器人伙伴作为聊天室的一个成员，表象上它和所有人是一样的。



但机器人就像是很多团队里都存在的那种个性员工，说的少做的多：

说的少——一个聊天室里，大部分时间机器人伙伴是沉默的，或者默默在后面做事的；说话的都是我们这些喜欢"闲扯"的人，真有事情来了，才想起机器人伙伴能不能帮忙给做了，这时候他才会被逼着跟你"说"两句。

做的很多——如果你愿意，机器人伙伴可以帮你做所有事。当然这里有一个度的问题，不是所有的事情都应该让机器人伙伴去做。机器人伙伴本质上是一种有规律下的封装，只有事情是稳定的、可持续的，才考虑招聘机器人来做。但是，千万不要无限的去招聘机器人，即使是私人的。因为和你招聘其他团队成员一样，想象一下你的团队无限扩展，有些方面自然有好处，但带来的问题也不言而喻。

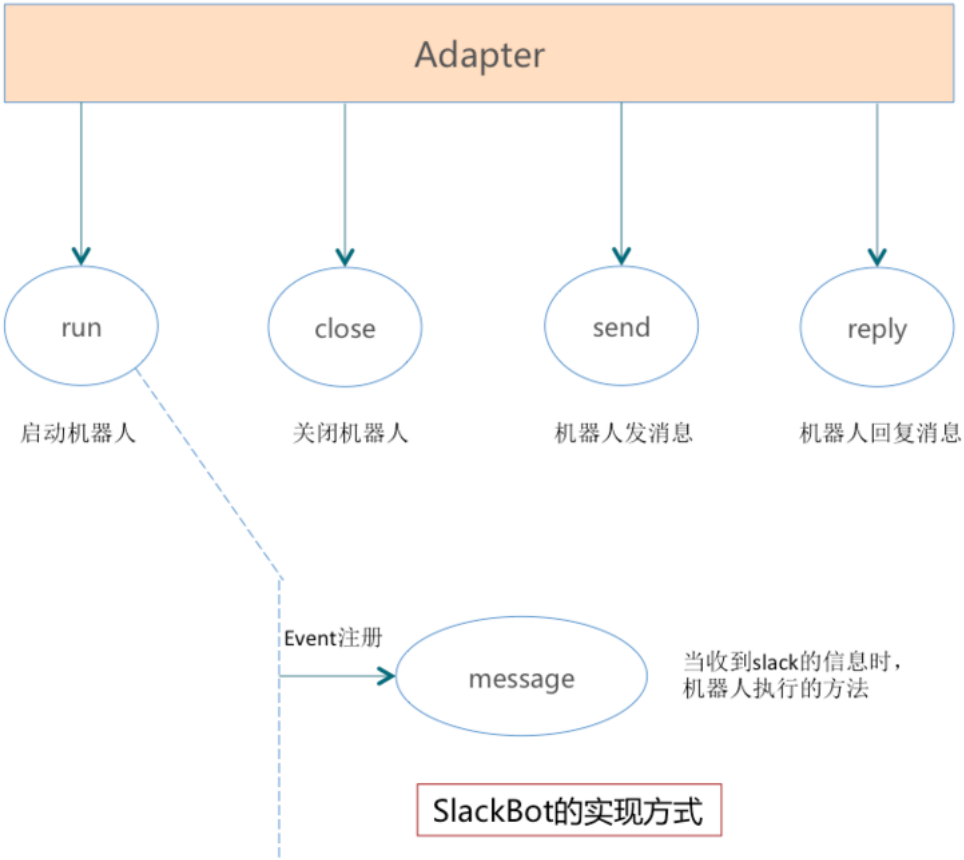
那一般我们怎么招聘机器人呢？再以 Hubot 举例，前面提到这是基于 CoffeeScript 的，需要一定的脚本基础，不过从我的使用情况来看（我脚本基础也很一般），关系也不大（具备 node，npm 相关的知识就可以），因为真正和 CoffeeScript 相关的工作很少，一般就两步（当然，这个是 Slack 适配后做了易用性，默认可不是这么简单的，后面会提到如何适配）：

```
if process.env.HUBOT_test
  regex = /test(z|s|ped)?\s*it/i
else
  regex = /test\s*it/i

robot.hear regex, (msg) ->
  # rd = httpclient.send()
  msg.send rd
```

1. **定义 robot**：每个机器人的定义方式基本上是一模一样的；
2. **匹配 command**：发送返回信息，上面只是截取的示例，一般会在匹配后，发送 http rest 请求实际去工作（这个就有很大的可操作空间了），将结果 format 后再发到聊天室中。

如果你的公司用的是没有与 Hubot 集成的 chat 软件，还需要做一次 client 的封装，这个稍有点复杂，需要一定的脚本基础，可参考 hubot-slack 项目：https://www.npmjs.com/package/@slack/client，我用一张图来说明 Hubot 的扩展架构，其集成时的插件点很明确（注：下图只标识了最重要的几个方法）：



通过实现 Adapter 的必要方法，完成机器人的生命周期管理。在与 Slack 集成时，稍有特殊性在于：run 方法中，注册了 Slack 的 message 事件（当 Slack 有消息时触发），在 message 方法里，通过消息类型、发送人、channel 等上下文信息，将具体消息封装后 dispatch 给机器人。

避免误区

我认为在接纳 ChatOps 这个理念的过程中，容易存在三种思想误区，会在一定程度上阻碍 ChatOps 的落地。

误区 1：ChatOps 纯粹是为了好玩。大家体验过人工智能，或者指挥过机器人做事，当时是持着什么样的心态去做的呢？我在一开始用 Hubot 的时候，兴冲冲的拉着部门同学分享，很直观的反馈就是：大家觉得很新

误区 2：聊天室里做事很不规范。企业用 IM，比如钉钉、Slack、HipChat，也有用微信、QQ 的，但很少有企业会把 IM 工具及内容纳入到标准管理体系中，很多时候就是纯粹的聊天工具。在这类工具中做事，大家会觉得无法保障规范性、可审计性等。

误区 3：Command 让工作不再专业。就像我们公司的产品 EOS（SOA 下的开发运行平台），自出生就饱受技术人员争议，原因是封装了太多底层实现。而 ChatOps 中的 Command 工作方式，同样会让我们觉得专业技能受到挑战。

上面这三种看法真的有道理吗？其实不然，在我看来这种误解的本质上来源于：

1. 不同层面上的认知偏差。很多同学都在广用开源和工具，但从来没有觉得这些屏蔽了底层实现，为何对于 ChatOps 中的机器人伙伴做事，就有这个感觉呢？比如说大家用 Eclipse 开发代码，很少有人关心 Eclipse 工具本身屏蔽的内容，但一旦在 Eclipse 加了些插件辅助，很多人就觉得不直观了；再比如很多前端同学使用 React、Bootstrap 这些框架，是否关心过它屏蔽了 prototype、闭包这些基础知识呢？这其实是不同层次的对问题的认知，说的直白些，我觉得是是惯性让人变得封闭，不想跳出习惯的工作方式。

2. 责任心的缺失，或者叫个人主义。在 ChatOps 领域中，我们都说要机器人，但有时候会发现团队里就你在贡献，这当然是个很不好的体验，让人很受打击；再者，聊天室里去工作，让新同学看着聊天窗口就能学到你的工作方法，这个会让一些人觉得不爽，仿佛侵犯了一些个人信息，他宁可去写文档或手把手的教，就是不想在一个好像被"监视"的环境下做事。这个其实涉及到了 Freedom & Responsibility 的问题，如何权衡相信大家自有评判。

总结

虽然接触 ChatOps 领域时间不长，但已深刻感受了其独特魅力：

1. **聊天室不再是聊天，随着伙伴角色的丰富与能力提升，聊天室会成为为一个协作、学习的基础支撑平台。**当然，不同于现在很多微课堂，这里会让你看到高手的实操方式、让每个成员可拥有很酷的机器人拍档。

2. **生态从小团队做起。**以前一说生态就被放得很大，即使企业里面说生态，也时常会放到基线平台的概念里；现在我们真的可以快速去建立小生态了，你只要在一些基础上招聘一些机器人，就可以为你的团队生态提供一份贡献，相信每个企业的可优化空间都很大吧。

3. **合理处理人与机器的关系**，不要再是 e-e 关系 ，而把他作为团队里的成员，最吃苦耐劳的成员来看待，这才是 ChatOps 所期望的。

4. 止于至善，这是我的校训，在 IT 这个领域，尤其是现在的 DevOps、ChatOps 领域更为适用，这些都是点滴积累，**精益求精的建设过程**，不可能有万能钥匙（标准产品）。

这里讨论的知识初步做一些基础运维的事情，但是正如文章之前所提，ChatOps 不局限于运维，后续的一些更高阶做法，会在团队实践后再做分享。

作者简介

顾伟，毕业于东南大学，现任普元公司主任架构师；先后参与和带领了华为 BME、中信银行 CBJUP、工商银行 CTP、中航信 RI、阿里云 ACE、普元云计算平台、普元 The Platform 等大型项目的交付；长期致力于 IT

如果你对 ChatOps、DevOps 相关话题感兴趣，欢迎扫码加入由顾伟主持的“普元云计算研发开放群”，讨论更多微服务、DevOps、容器相关内容，加群暗号“ChatOps”。

感谢木环对本文的审校。

给 InfoQ 中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（@InfoQ，@丁晓昀），微信（微信号：InfoQChina）关注我们。

语言 & 开发 架构 运维

0 人喜欢

收藏 评论 微信 微博

写下你的想法，一起交流

发表评论

注册/登录 InfoQ 发表评论

注册/登录