# Cloudfoundry系统

## 概要

CloudFoundry使用Ruby语言进行编写和实现。主要包含Router、CloudController、NATS、DEA、Stager、HealthManager、blobstore、ServiceGateway、ServiceNode等组件，消息为通信核心，协调各个组件模块的运行。扩展Nginx，实现网络请求负载均衡。通过服务网关模式管理cloudfoundry内部的MySQL、Redis、Rabbit等扩展服务。支持Ruby、Java等多语言的应用，目前包含了JDK 5、JDK7等运行环境，内嵌了支持Java的 Web服务器 Tomcat等。

## 架构图



## CloudFoundry启动过程

运行如下命令：vcap-master\dev\_setup\bin\vcap\_dev start

启动各个模块：Router、NATS、CloudController、DEA、ServiceGateway、ServiceNode。

### Router启动

获取IP参数

启动Thin server

连接NATS

订阅消息

向系统注册Router自己

设置vars和sweeper

定期发布一个Router已启动的消息

### CloudController启动

CloudController是一个Ruby on Rails程序，使用MVC架构。

### DEA启动

Droplet Execution Agency

dea.rb 中生成 agent 对象，调用agent run方法进行初始化。agent.rb中run方法处理内容如下：

初始化状态

订阅消息

|  |
| --- |
| *# Setup our listeners..*  NATS.subscribe('dea.status') { |msg, reply| process\_dea\_status(msg, reply) }  NATS.subscribe('droplet.status') { |msg, reply| process\_droplet\_status(msg, reply) }  NATS.subscribe('dea.discover') { |msg, reply| process\_dea\_discover(msg, reply) }  NATS.subscribe('dea.find.droplet') { |msg, reply| process\_dea\_find\_droplet(msg, reply) }  NATS.subscribe('dea.update') { |msg| process\_dea\_update(msg) }  NATS.subscribe('dea.stop') { |msg| process\_dea\_stop(msg) }  NATS.subscribe("dea.#{uuid}.start") { |msg| process\_dea\_start(msg) }  NATS.subscribe('router.start') { |msg| process\_router\_start(msg) }  NATS.subscribe('healthmanager.start') { |msg| process\_healthmanager\_start(msg) }  NATS.subscribe('dea.locate') { |msg| process\_dea\_locate(msg) } |

恢复已存在应用的状态

定期发送状态消息

发布已启动DEA的消息

|  |
| --- |
| NATS.publish('dea.start', @hello\_message\_json)  send\_advertise |

## 应用发布过程

以vmc push为例

1.vmc 会调用lib\mothership.rb的start方法。

mothership.rb位于D:\reference\PAAS\tools\Ruby193\lib\ruby\gems\1.9.1\gems\mothership-0.3.5\lib

2. \lib\mothership.rb的start方法

## 模块结构

### NATS

消息服务处理中心，基于EventMachine开源项目和Reactor模式。

## MCF目录结构

/var/vcap/sys/log：存放日志信息

/var/vcap/data/packages/：存放运行CloudFoundry的源代码

/var/vcap/data/jobs/：

/var/vcap/data/jobs/micro/83.1-dev/templates/cloud\_controller

/var/vcap/data/jobs/micro/83.1-dev/packages： 目录链接，指向/var/vcap/data/packages/

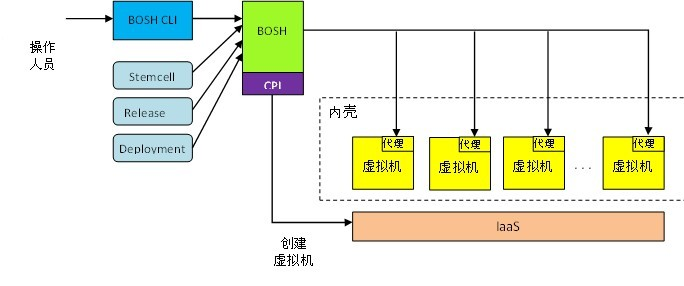
/var/vcap/packages/cloud\_controller：目录链接，指向/var/vcap/data/packages

## VMC

通过VMC进行的操作实际上到最后都是CC在处理。VMC 与 CC 连接的桥梁：[Gem rest\_client](https://github.com/adamwiggins/rest-client" \t "_blank)

# BOSH

## 结构图



BOSH安装的基本内容：

一个作为虚拟机模板的stemcell、一个作为待部署软件的 Release，以及一个用来定义部署配置的部署清单文件。

BOSH安装基本步骤：

下载、上传虚拟机模板的stemcell

创建、上传一个作为待部署软件的 Release

准备部署清单文件

根据部署清单文件进行发布

通过BOSH安装多节点、分布式的cloudfoundry时，需要有如下条件：

充足的IP

充足、稳定的内存、硬盘、CPU等硬件

稳定的网络

需要操作的内容：

用vCenter对硬件资源分配

NFS Server

网关路由

BOSH历史数据清理

## BOSH的机制

BOSH 视作一台负责协调分布式系统部署过程的服务器或机器人。有个Ruby 工具可以与 BOSH 命令行界面 (CLI) 进行交互。BOSH 需要以下三个必备项才能开始部署系统：一个stemcell、一个Release（要安装的软件）和一个部署清单(Deployment Manifest)。

Stemcell：在云平台中，虚拟机通常是从模板克隆而来的。一个stemcell就是一个包含标准 Ubuntu Linux的虚拟机模板。该模板中还嵌入了一个 BOSH 代理，以便 BOSH 可以控制从该stemcell克隆出来的虚拟机

Release：Release包含若干组将要安装到目标系统上的软件代码和配置。每个虚拟机上都要部署一组软件，这组软件称作一个作业（job）。

部署：部署就是使静态的Release变成虚拟机上可运行的软件的过程。部署清单定义了部署所需的实际参数值。在部署过程中，BOSH 会替换掉Release中的参数，从而使软件按照我们规划的配置来运行。

当上述 3 项内容都准备好后，BOSH CLI 工具会将它们上传到 BOSH。接着，用BOSH 来安装分布式系统包括以下主要步骤：

1) 如果Release中的某些包需要编译，BOSH 首先会创建几个临时虚拟机（worker，工作者虚拟机）来编译它们。编译完后，BOSH 便会销毁这些工作者虚拟机，将所产生的二进制代码存储在其内部blobstore中。

2) BOSH 创建一个虚拟机池，池中的虚拟机将成为该Release要部署到的节点。这些虚拟机是从装有 BOSH 代理的stemcell克隆而来的。BOSH使用CPI接口调用vSphere的虚拟机创建操作API，来自动化的完成虚拟机的创建和配置工作。CPI接口同样适合于OpenStack、AWS等其他IaaS管理平台。

3) 对于该Release的每个作业，BOSH 会从该池中选取一个虚拟机，然后根据部署清单更新该虚拟机的配置。具体的配置可以包括 IP 地址、持久磁盘的大小等。

4) 重新配置完该虚拟机后，BOSH 会向每个虚拟机内的代理发送命令。这些命令通知该代理安装软件包。在安装期间，该代理可能会从 BOSH 下载包并安装它们。安装完毕后，该代理会运行启动脚本来启动该虚拟机的作业。

5) BOSH 重复执行第 3 步至第 4 步，直至所有作业都部署完毕并启动为止。这些作业可以同时部署，也可以按顺序部署。清单文件中的“max\_in\_flight”值用于控制并行部署的作业数。当该值为 1 时，表示这些作业逐一按顺序部署。对于较慢的系统，该值有助于避免因资源拥塞而造成的超时。该值大于 1 时，表示这些作业可以并行部署。

# CF单节点的基于源码安装

涉及到域名设置、SSH通道建立、不能使用高版本的VMC、只能使用vmc -v 0.3.18

# CF新服务的建立

用Ruby基于CF 服务系统的结构写Node和Gateway，并用source进行联编。

cloudfoundry的文档中没有提到的设置：

在gateway.yml文件中的service里必须加上参数timeout。