Open Source. Performant. Simple. Scalable.

A central nervous system for modern, reliable,
and scalable cloud and distributed systems.

关于我

宋飞

2008年10月 福冈 EC方面开发和维护 PHP

2015年 1月 东京 B2B市场营销平台开发 Scala Perl

云消息系统 NATS

nats 是一个轻量级的云消息系统，目前提供了使用 Go 开发的服务器版本以及多种编程语言的客户端开发包。NATS是一个基于事件驱动的、基于发布和订阅模型的轻量级消息系统。它基于EventMachine实现。

网络编程和并发编程的框架 EventMachine

EventMachine 是一个基于Reactor设计模式的、用于网络编程和并发编程的框架。Reactor模式描述了一种服务处理器，它接受事件并将其分发给已注册的事件处理。这种模式的好处就是清晰的分离了时间分发和处理事件的应用程序逻辑，而不需引入多线程来把代码复杂化。

What NATS is..

✓ High-Performance

✓ Always on and available

✓ Extremely light-weight

✓ Fire and Forget - At Most Once

✓ Pub/Sub

✓ Distributed Queues

✓ Request/Reply

What NATS is NOT..

✓ Enterprise Messaging System

✓ Persistence

✓ Transactions

✓ Enhanced Delivery Models

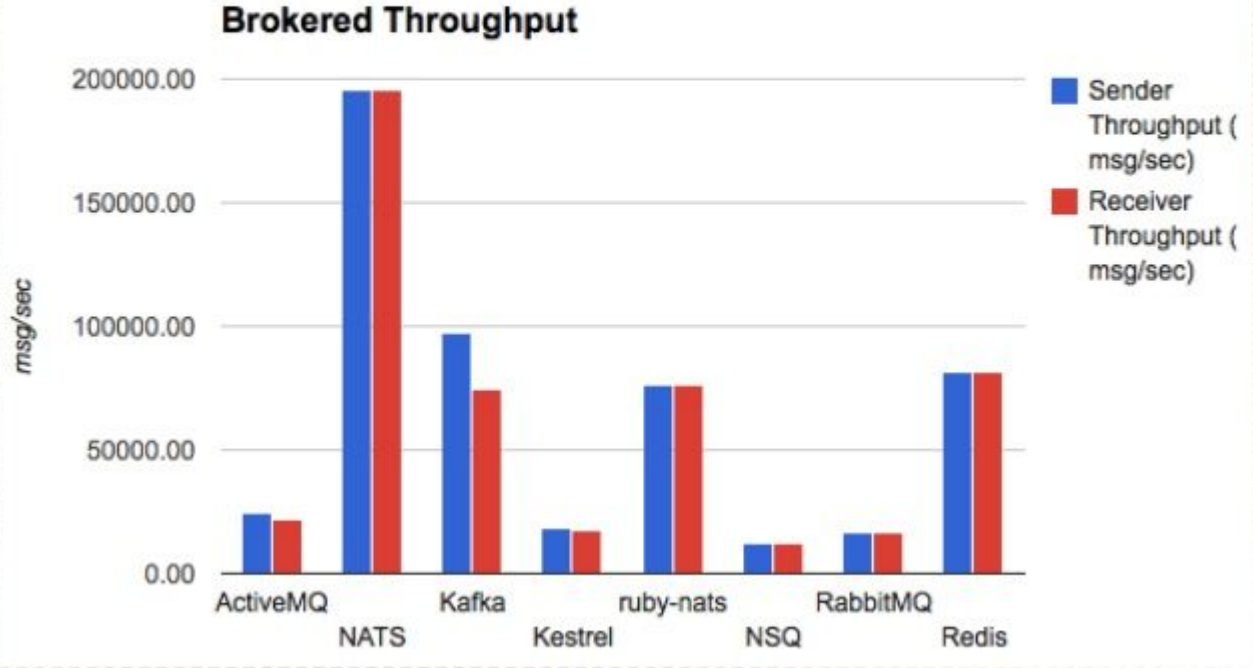
✓ Queueing Product

Performance

- Originally written to support CloudFoundry
- In use by CloudFoundry, HTC, Baidu, Apcera and others
- Written first in Ruby -> 150k msgs/sec
- Rewritten at Apcera in Go (Client and Server)
- First pass -> 500k msgs/sec
- **Current Performance -> 8M msgs/sec**

Performance 4k payloads

Courtesy - http://www.bravenewgeek.com/dissecting-message-queues/



NATS Server

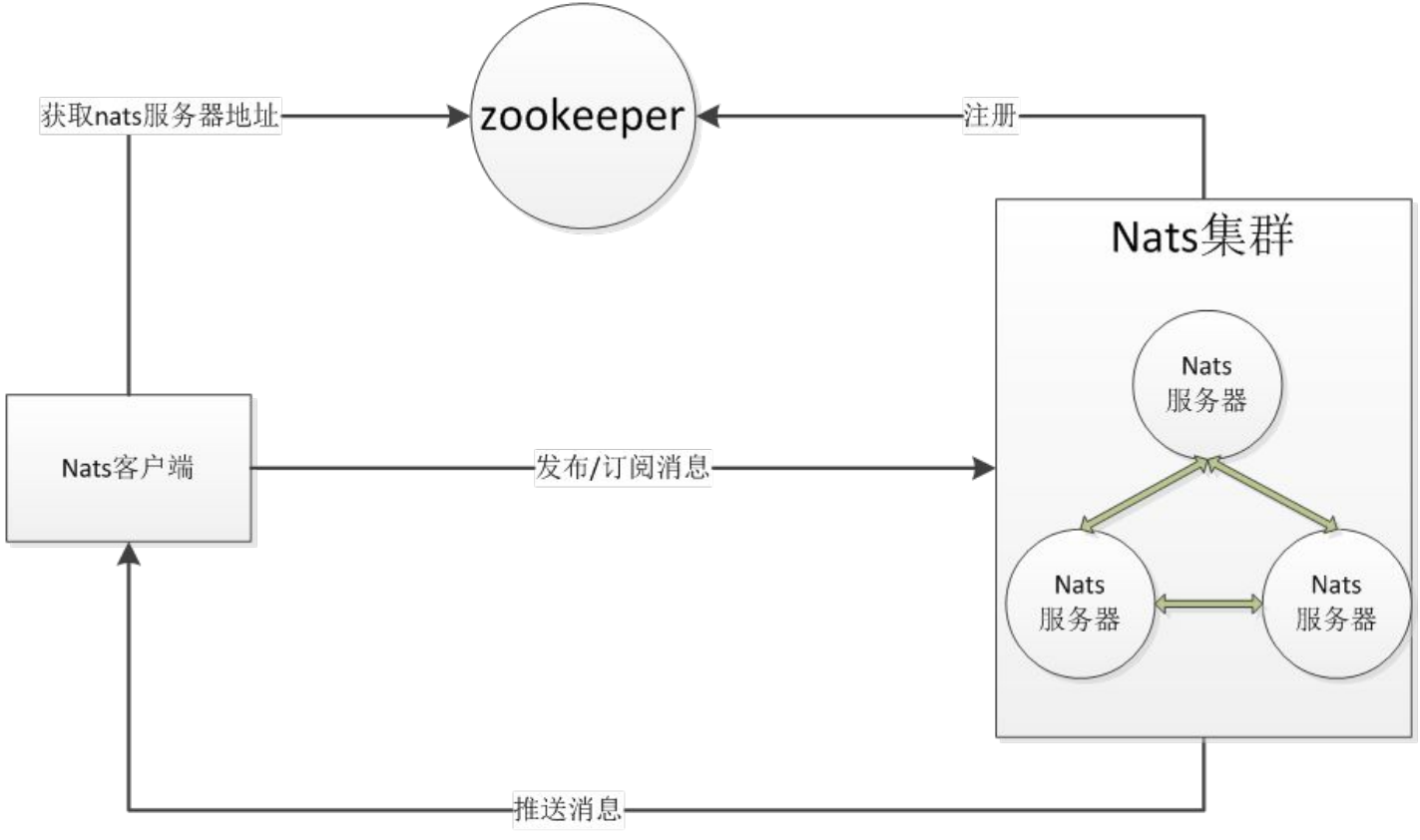
NATS provides a lightweight server that is written in the Go programming language. Apcera actively maintains and supports the NATS server source code, binary distributions, and Docker image.

NATS Client

There are several client libraries for NATS. Apcera actively maintains and supports the Go, Node, and Ruby clients, and there are several community-provided clients.

You can write your own client in any language you choose. NATS provides a simple, text-based protocol that makes writing clients a breeze.

京东云擎NATS集群化方案设计(link)



```
[vagrant:~]$ docker run -d -p 4222:4222 --name nats-main nats
c573396c80239dab5a52c16d71fb719870ff05eefd6bfea61a60c71dca5dcce9
[vagrant:~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CRE
c573396c8023        nats:latest         "/gnatsd -c /gnatsd.   3 s
```

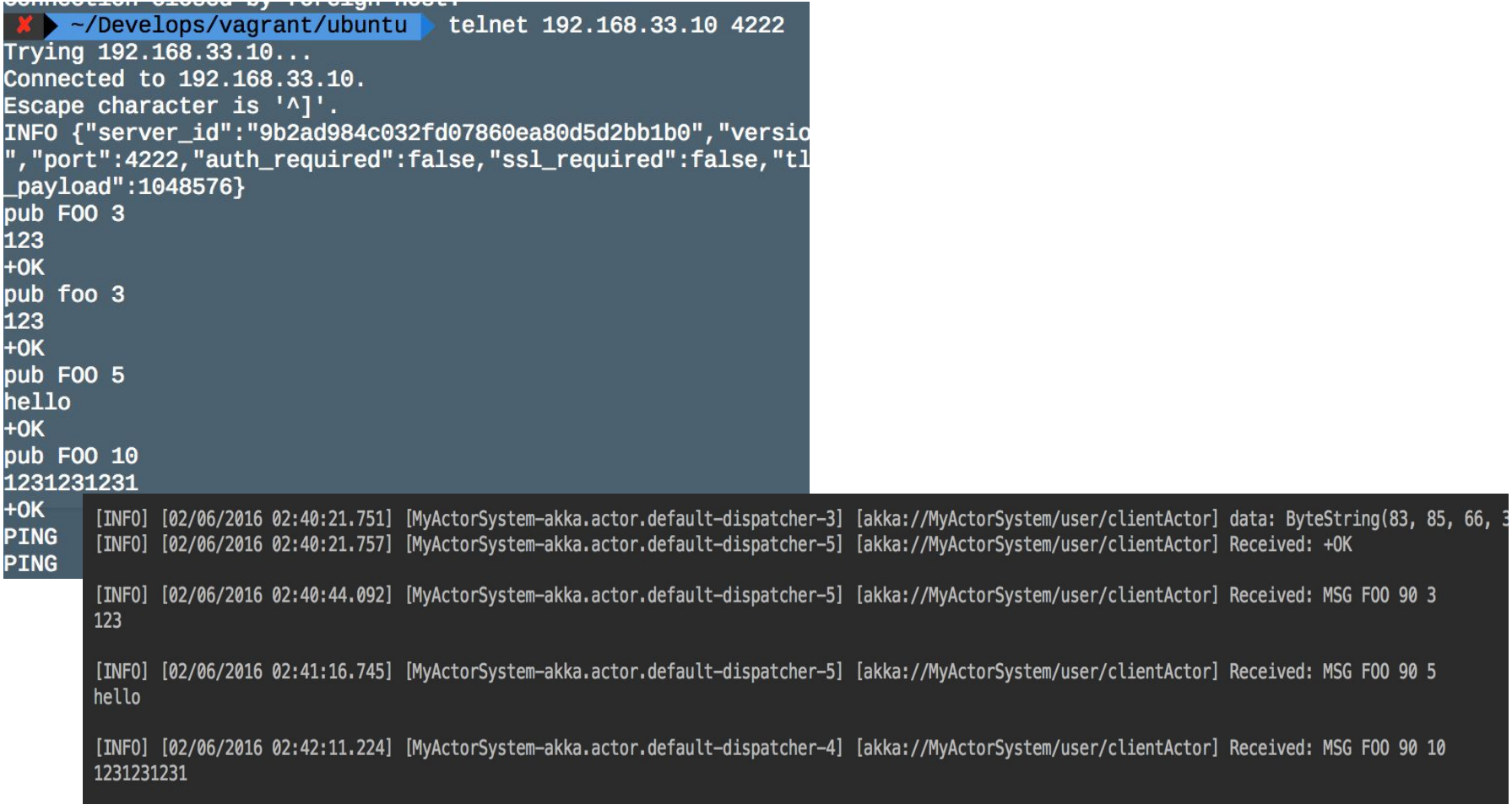
build.sbt × Hello.scala ×

```scala
package com.example
import akka.actor._
import akka.io.{ IO, Tcp }
import akka.util.ByteString
import java.net.InetSocketAddress

object Hello {
  def main(args: Array[String]): Unit = {
    println("Hello, world!")

    val system = ActorSystem("MyActorSystem")
    val clientActor = system.actorOf(Client.props(new InetSocketA
    Thread.sleep(500L)
    clientActor ! ByteString("SUB FOO 90\r\n")
    system.awaitTermination()
  }
}

object Client {
  def props(remote: InetSocketAddress) =
    Props(classOf[Client], remote)
}
```


~/Develops/vagrant/ubuntu telnet 192.168.33.10 4222
Trying 192.168.33.10...
Connected to 192.168.33.10.
Escape character is '^]'.
INFO {"server_id":"9b2ad984c032fd07860ea80d5d2bb1b0","versio
","port":4222,"auth_required":false,"ssl_required":false,"tl
_payload":1048576}
pub FOO 3
123
+OK
pub foo 3
123
+OK
pub FOO 5
hello
+OK
pub FOO 10
1231231231
+OK
PING
PING
[INFO] [02/06/2016 02:40:21.751] [MyActorSystem-akka.actor.default-dispatcher-3] [akka://MyActorSystem/user/clientActor] data: ByteString(83, 85, 66, 3
[INFO] [02/06/2016 02:40:21.757] [MyActorSystem-akka.actor.default-dispatcher-5] [akka://MyActorSystem/user/clientActor] Received: +OK
[INFO] [02/06/2016 02:40:44.092] [MyActorSystem-akka.actor.default-dispatcher-5] [akka://MyActorSystem/user/clientActor] Received: MSG FOO 90 3
123
[INFO] [02/06/2016 02:41:16.745] [MyActorSystem-akka.actor.default-dispatcher-5] [akka://MyActorSystem/user/clientActor] Received: MSG FOO 90 5
hello
[INFO] [02/06/2016 02:42:11.224] [MyActorSystem-akka.actor.default-dispatcher-4] [akka://MyActorSystem/user/clientActor] Received: MSG FOO 90 10
1231231231

云消息系统 NATS

http://nats.io

Akka 2.4.1 TCP

http://doc.akka.io/docs/akka/2.4.1/scala/io-tcp.html

EventMachine入門

http://keijinsonyaban.blogspot.jp/2010/12/eventmachine.html