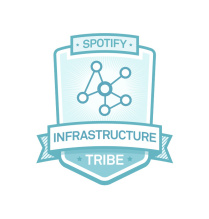
# Spotify的后端基础架构

发表于 2013年3月15日[Gösta Forsum](https://labs.spotify.com/author/gostaforsum/" \o "Posts by Gösta Forsum)

[](https://spotifylabscom.files.wordpress.com/2013/03/spinfratribe1.jpeg)

在这篇博客我会给大家展示一下我们是如何搭建Spotify的后端架构的。我们的后端架构建做了非常多的工作 - 在某些领域我们已经走过了漫长的道路，在有的地方，我们刚刚起步。

为了更好的理解我们为什么搭建这些基础设施，我们需要了解一下的Spotify的开发团队的组织原理。目前，我们大约有300名工程师在Spotify - 并且我们正在迅速增长。

背景

成长 - 在Spotify,业务迅速的迭代变化着。每天的用户数，后端的节点数，我们的客户端运行的硬件平台的数量，正在研发的产品的团队数量，我们平台上正在运行的App的数量，我们曲库中拥有的歌曲的数量。

速度 - 随着我们的成长，我们必须密切注意着 那些使我们开发速度减慢的因素。我们下大力气消除团队之间的依赖关系，并从我们的架构中移除不必要的复杂性。

自主团队 -在Spotify的一个重要的概念是每个开发团队都应该是自主的。每一个开发小组都应该是能够独立的执行任务。即使有两个开发部门之间有业务依赖关系，他们也能找到各自合适的方式来前进。为了确保每个小组都能取得进展，即使他们和别的组有依赖关系，我们在Spotify内部执行这些策略：我们的透明代码模型（transparent code）和我们的自助服务架构(self service)。

**透明代码模型 (transparent code)**- 所有Spotify的代码对 所有的开发者 在透明代码模型下都是可见的。这就意味着，在Spotify的客户端，Spotify的后端和Spotify的服务架构端的所有的代码，对于Spotify的程序员来说，都是可以阅读和修改的。如果一个小组正在修改一部分代码，但是这部分代码正在被其他团队使用，他们总是有一些方式先提交自己的修改然后继续他们的开发任务。

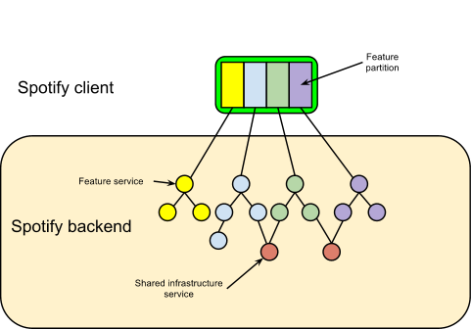
在实践中，Spotify的透明代码模型的工作原理是所有团队共享同一个集中式的Git服务器。每个git repo 都会有一个专门的系统所有者是负责代码的，并确保它不会rot。透明代码模型可以确保每个人都可以推进项目进度，而且每个人都有得到其他人的代码。这使得Spotify的code 库一直进化，给程序员一个积极和开放的工作环境。

**自助服务基础设施（**self service**）** - 所有的基础应该组成一个有机的整体还发挥作用。这样的话，没有必要等待其他球队获得硬件资源，设置一个存储集群或做配置修改。Spotify的后端基础服务架构是由 硬件和软件的几层组成的，从物理机器到消息机智和存储解决方案组成。

**开源** - 我们尽可能的使用开源工具。我们在后端用了很多的软件开保证我们的核心服务，Spotify正在持续的扩大这些软件性能的极限。所以我们给开源社区做出了很多贡献 例如 Apache 的Cassandra和ZMQ。我们几乎没有非开源软件，因为我们不相信他们会能迅速的满足我们的日益增长的用户需求。

**文化** - 在Spotify，我们坚信个人的主观能动性。这一点在自主团队组织上反应的淋漓尽致。对于工程师来说，有很多可能性在Spotify其他领域内尝试合作，以确保每个人都保持对工作充满热情。在我们的hackdays，工程师们可以尝试他们有的各种新奇的想法。

**组织结构**

[](https://spotifylabscom.files.wordpress.com/2013/03/sparch.png)

Spotify的架构在几种不同的划分方式。首先，通过功能分区。稍微简单化的描述是，所有的页面和意见，我们的客户的物理屏幕区域被一些小组拥有。

所有的在Spotify的客户端的功能属于其他特定小组去服务。

小队负责在所有平台上的功能 - 从它如何出现在iOS设备中或通过Spotify的后端到面向批处理的数据处理，发生在我们的Hadoop集群功率处理的实时请求的浏览器上一路功能，如建议，电台和搜索。

如果其中一个功能出现故障，客户的其他功能是独立的，并会继续工作。如果存在特征之间的弱相关性，一个特征的失败有时可能导致另一特征的服务质量的下降，而不是对整个Spotify的服务失败。

由于所有的用户不使用所有的功能，同时，具有由特定特征的后端处理的用户的数目通常比整个Spotify的服务的用户的数量少得多。

由于各地的一条具体特征的知识集中于一个球队是非常容易的A / B测试功能，看看收集到的数据，并采取与涉及的所有相关人员作出明智的决定。

特征分区给出的可扩展性，可靠性和聚焦团队努力的一个有效的途径。

# 后端基础结构

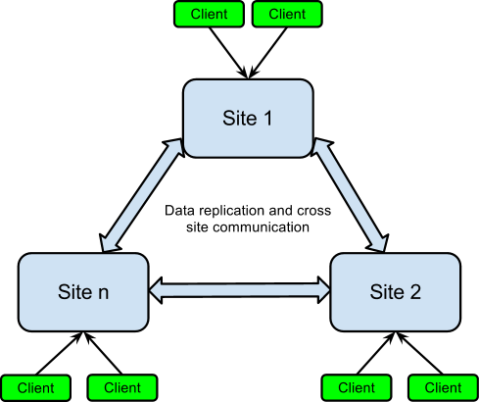
按功能划分我们的问题，并给予高度熟练的跨职能小队任务照顾，并与该功能下班后，现在的问题变成，我们如何建立基础设施，有效地支持了班长？

我们怎样才能确保球队能够发展以极快的速度它们的特点，而不用担心被阻止其他球队？我们的基础设施如何解决全球各地的结垢难题？我已经谈到了我们透明的代码模型，总是可以让一支球队前进，但也有除了功能开发小队组织的其他部分。

在许多组织中，你有利用数据库​​及其架构保健数据库管理员，你通常需要经历一个运营部门获得数据中心分配的硬件等组织中的这些特殊功能成为瓶颈的时候同时有100个队苛求他们的服务。为了解决这个问题，我们正在开发在Spotify的后端基础设施，是完全独立的服务。完全自助服务意味着任何球队可以开始开发，而不必与组织的其余部分进行交互的环境中生活服务迭代。

为了实现这一目标，我们需要解决的范围在几个不同的区域的问题。我会在这里介绍几个重要的。

配置 - 在开发新功能的阵容通常需要在多个地点部署此服务。我们正在建设的基础设施，使小队自行决定是否服务应在Spotify的自己的数据中心，或者如果该功能被部署可以使用公共云产品。Spotify的基础设施力求以最小化在我们自己的数据中心和在公共云中运行之间的差异。总之你得到更好的延迟，在我们自己的数据中心更稳定的环境。在公有云，你得到的硬件更快的配置和更动态扩展的可能性。

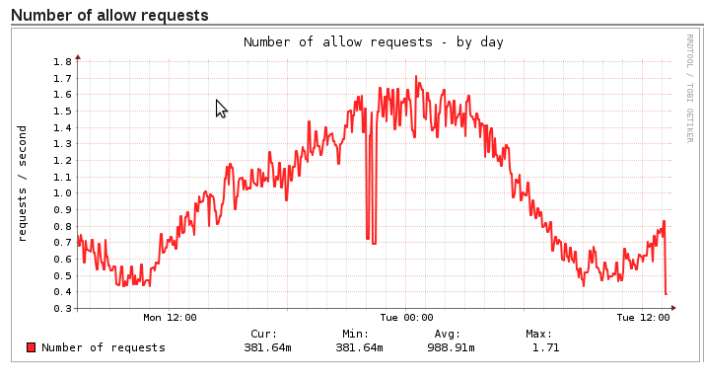
[](https://spotifylabscom.files.wordpress.com/2013/03/siterepl.png)

连接到他们最亲近的数据中心客户的Spotify。

存储 - 大多数功能需要某种形式的存储，明显的例子是播放列表和“跟随”功能。建立了数以百万计的人将使用功能的存储解决方案不是一件容易的事，这里面有很多事情必须考虑：访问的模式，网站，能力，一致性，备份，退化的情况下之间进行故障转移网站等之间的净分裂没有履行一个通用的方式，所有这些要求没有简单的方法。每个功能的阵容将不得不创建一个适合特定业务需求的存储解决方案。Spotify的基础设施提供存储一些不同的选择：卡桑德拉，PostgreSQL和memcached的。

如果需要的特征数据进行分区，那么球队必须实现其业务的拆分本身，然而，许多服务依赖于卡桑德拉做站点之间数据的完整副本。设置了复制和站点之间的故障切换一个完整的存储集群是复杂的，所以我们正在建设基础设施安装和维护多站点卡桑德拉或PostgreSQL集群作为一个单元。对于人们对Spotify的API开发应用程序将有一个存储作为不需要任何集群的任何设置一个服务选项。存储作为服务选项将被限制在一个非常简单的键值存储。

消息 - Spotify的客户端和后端服务进行通信使用以下范式：请求 - 应答，短信和发布订阅。我们已经建立了自己的低时延，低开销的消息层，并计划将其与高送保障，故障切换路由和更先进的负载均衡扩展。

容量规划 - Spotify的的增长带动了大量的流量到后端。每个队必须确保其功能一直扩展到当​​前负载。小队可以选择手动通过监测流量的服务来跟踪这一点，并找出并解决瓶颈和向外扩展需要。我们也建立一个基础架构，允许队与负载自动调整他们的服务。自动缩放通常只适用于你知道的瓶颈，所以总有人工监控的球队需要处理的一定水平。我们的基础设施允许轻松创建图表和警报，以支持这一点。[](https://spotifylabscom.files.wordpress.com/2013/03/graph.png)

其他服务绝缘 - 随着新功能和服务的开发，他们倾向于互相调用在不平凡的方式。这是很重要的是所有的小队觉得他们能以全速运行，同时尽量减少Spotify的其他部分产生不利影响的风险。为了避免这种情况，我们的消息传输层具有速率限制和权限的系统。速率限制有一个默认的阈值 - 这使得小队调用其他服务，来试图想办法。如果异常繁重的交通预计，敢死队将需要协调并同意如何共同处理这个问题。不同的功能总是在不同的服务器或虚拟机上运行，​​以避免有一个行为不端的服务拿下另一个。

正如我在这篇文章开头所提到的，很多，这是正在进行中的工作，也有很多的未来我们的方式非常有趣的挑战。我在这里提出的观点代表了我们如何现在看到Spotify的事情快照，因为我们都沉迷于持续改进，明天我们可能已经改变了一些东西......

当然，如果你觉得这是有趣的，看看我们的[未平仓头寸](http://www.spotify.com/us/jobs/view/oGaNWfwj/" \t "_blank)。

Measure

Measure