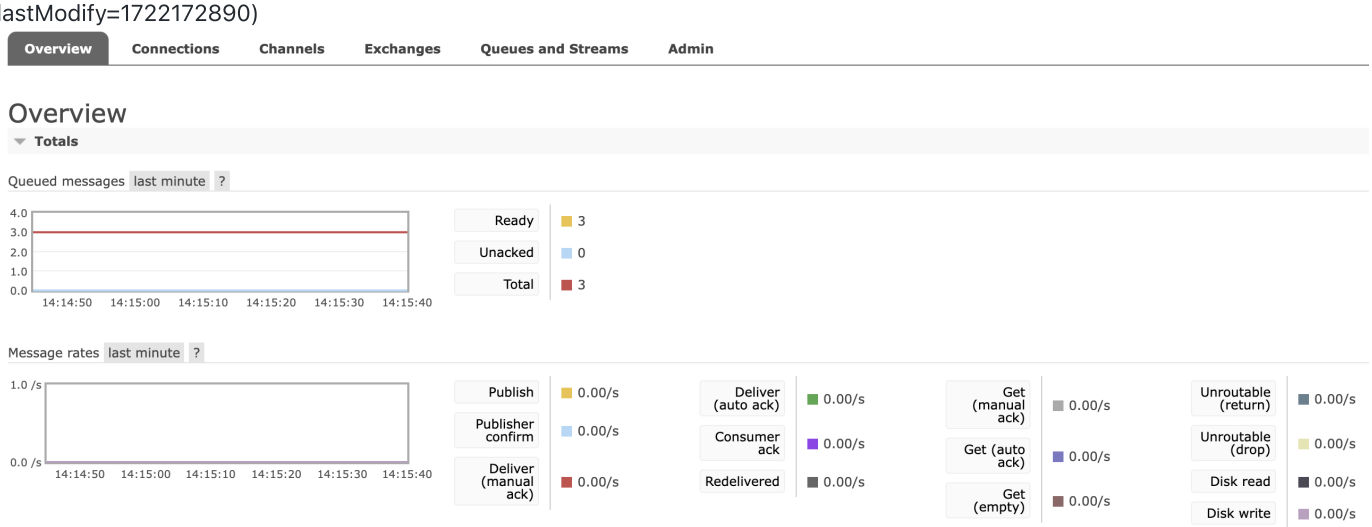
RabbitMQ集群机制及企业实践

-- 楼兰

这一章节主要介绍RabbitMQ在企业实践中保证RabbitMQ数据安全的一些常用措施。

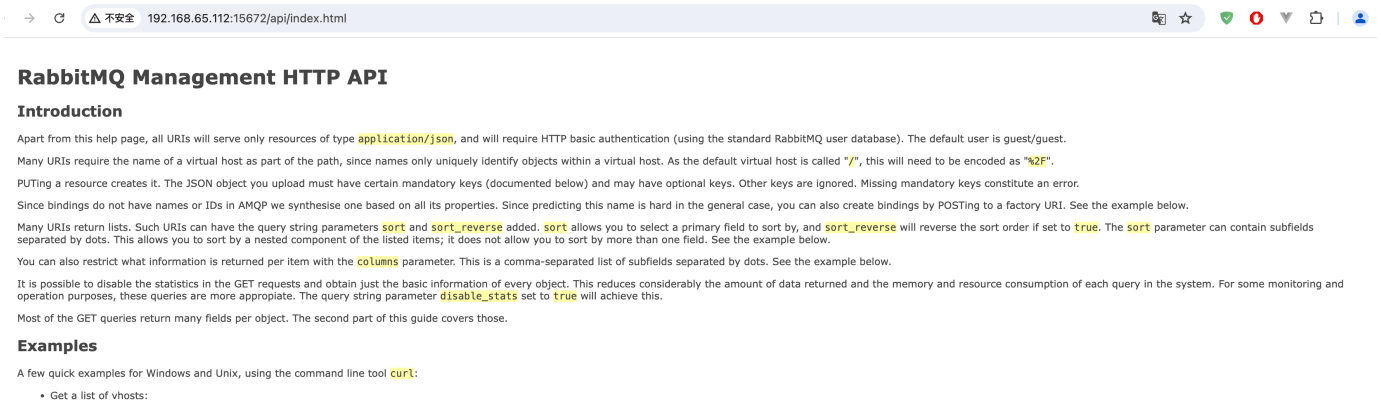
一、RabbitMQ的性能监控

关于RabbitMQ的性能监控，在管理控制台中提供了非常丰富的展示。例如首页这个整体监控页面，就展示了非常多详细的信息。。



还包括消息的生产消费频率、关键组件使用情况等等非常多的信息，都可以从这个管理控制台上展现出来。但是，对于构建一个自动化的性能监控系统来说，这个管理页面就不太够用了。为此，RabbitMQ也提供了一系列的HTTP接口，通过这些接口可以非常全面的使用并管理RabbitMQ的各种功能。

这些HTTP的接口不需要专门去查手册，在部署的管理控制台页面下方已经集成了详细的文档，我们只需要打开HTTP API的页面就能看到。



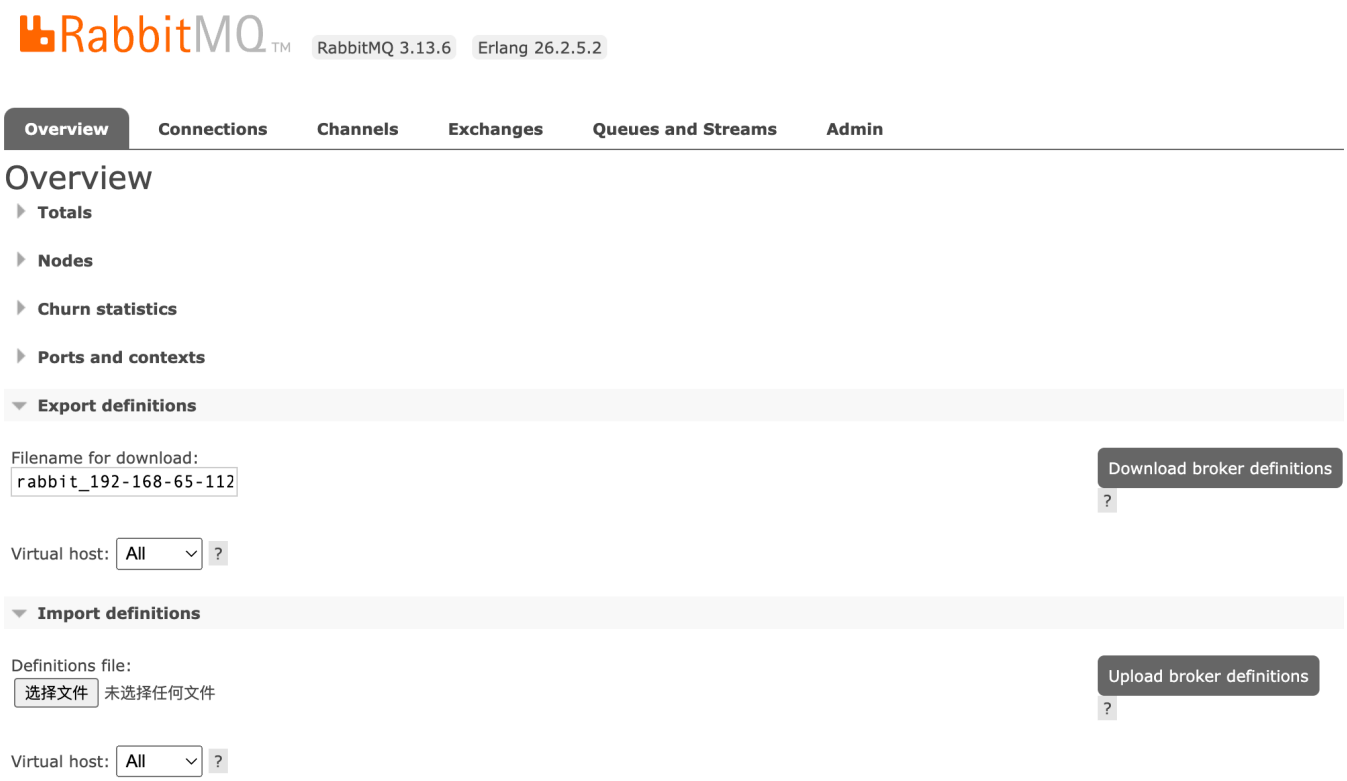
比如最常用的 http:///[server:port]/api/overview 接口，会列出非常多的信息，包含系统的资源使用情况。通过这个接口，就可以很好的对接Promethus、Grafana等工具，构建更灵活的监控告警体系。

可以看到，这里面的接口相当丰富，不光可以通过GET请求获取各种消息，还可以通过其他类型的HTTP请求来管理RabbitMQ中的各种资源，因此在实际使用时，还需要考虑这些接口的安全性。

二、RabbitMQ的备份与恢复

RabbitMQ有一个data目录会保存分配到该节点上的所有消息。`我们的实验环境中，默认是在/var/lib/rabbitmq/mnesia目录下` 这个目录里面的备份分为两个部分，一个是元数据(定义结构的数据)，一个是消息存储目录。

对于元数据，可以在Web管理页面通过json文件直接导出或导入。



对于消息，可以手动进行备份恢复

其实对于消息，由于MQ的特性，是不建议进行备份恢复的。而RabbitMQ如果要进行数据备份恢复，也非常简单。

首先，要保证要恢复的RabbitMQ中已经有了全部的元数据，这个可以通过上一步的json文件来恢复。

然后，备份过程必须要先停止应用。如果是针对镜像集群，还需要把整个集群全部停止。

最后，在RabbitMQ的数据目录中，有按virtual hosts组织的文件夹。你只需要按照虚拟主机，将整个文件夹复制到新的服务中即可。持久化消息和非持久化消息都会一起备份。 `我们实验环境的默认目录是` /var/lib/rabbitmq/mnesia/rabbit@192-168-65-193/msg_stores/vhosts

三、使用联邦插件进行远程消息同步

1、插件的作用

在企业中有很多大型的分布式场景，在这些业务场景下，希望服务也能够同样进行分布式部署。这样即可以提高数据的安全性，也能够提升消息读取的性能。例如，某大型企业，可能在北京机房和长沙机房分别搭建RabbitMQ服务，然后希望长沙机房需要同步北京机房的消息，这样可以让长沙的消费者服务可以直接连接长沙本地的RabbitMQ，而不用费尽周折去连接北京机房的RabbitMQ服务。这时要如何进行数据同步呢？搭建一个跨度这么大的内部子网显然就不太划算。这时就可以考虑使用RabbitMQ的Federation插件，搭建联邦队列Federation。通过Federation可以搭建一个单向的数据同步通道。

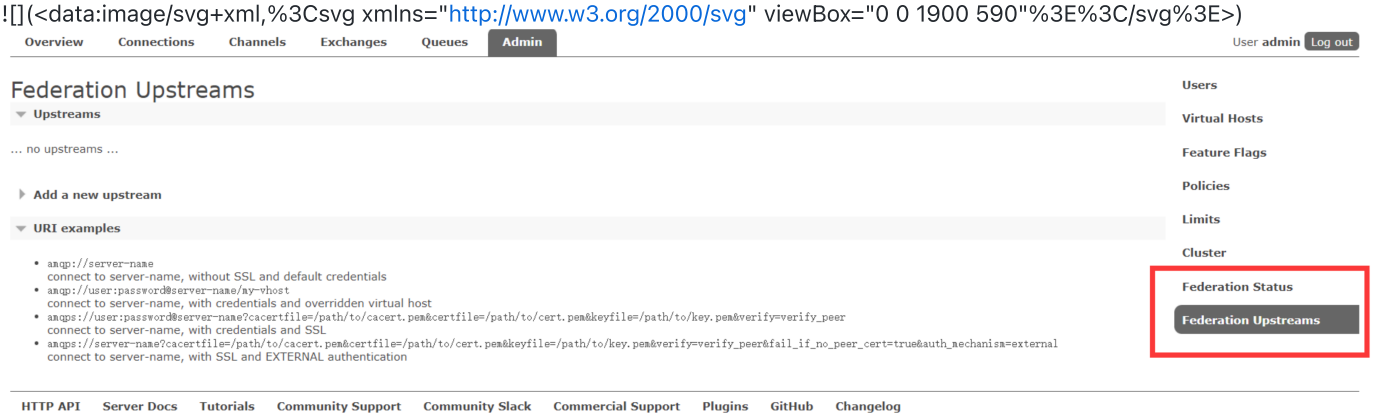
2、使用步骤

1、启动插件

RabbitMQ的官方运行包中已经包含了Federation插件。只需要启动后就可以直接使用。

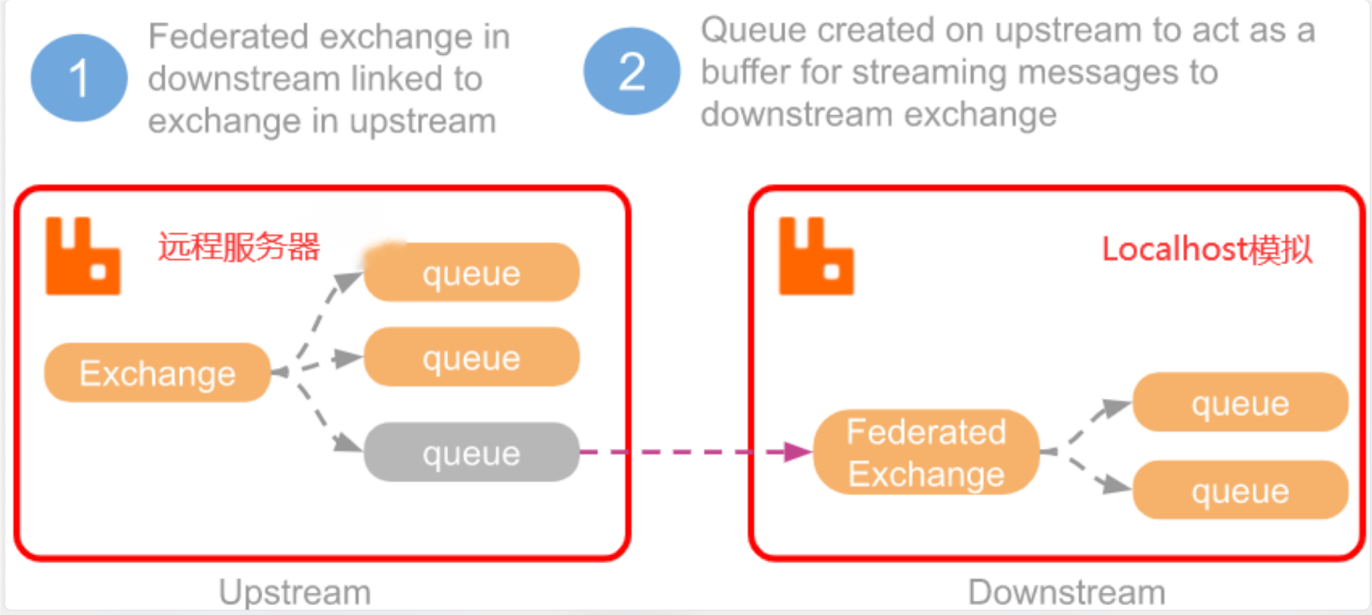
```
# 确认联邦插件
rabbitmq-plugins list|grep federation
[  ] rabbitmq_federation               3.13.6
[  ] rabbitmq_federation_management    3.13.6

# 启用联邦插件
rabbitmq-plugins enable rabbitmq_federation
# 启用联邦插件的管理平台支持
rabbitmq-plugins enable rabbitmq_federation_management
```

插件启用完成后，可以在管理控制台的Admin菜单看到两个新增选项 Federation Status和Federation Upstreams。



2、配置Upstream

Upstream表示是一个外部的服务节点，在RabbitMQ中，可以是一个交换机，也可以是一个队列。他的配置方式是由下游服务主动配置一个与上游服务的链接，然后数据就会从上游服务主动同步到下游服务中。

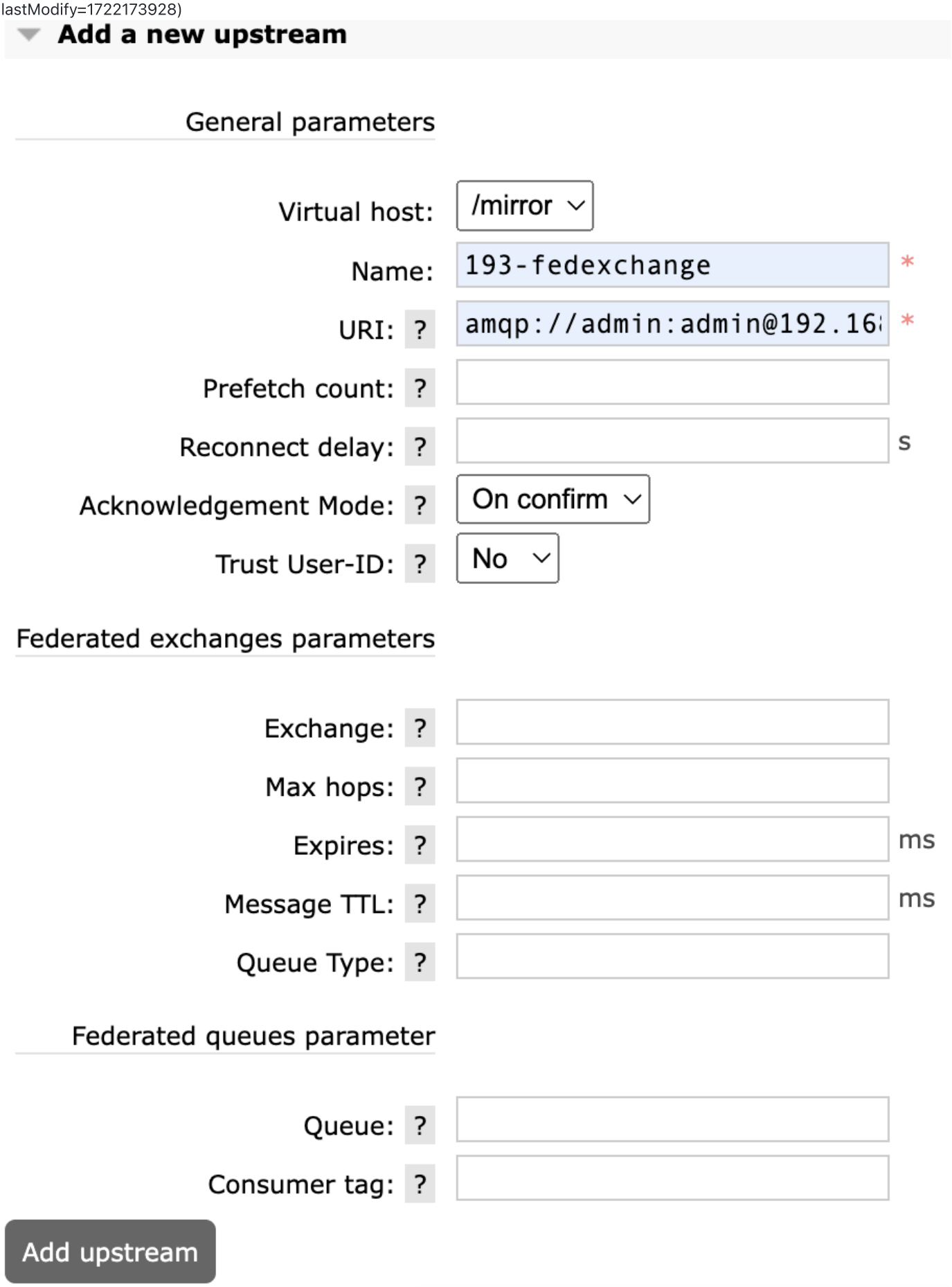


接下来我们用192.168.65.112上的的RabbitMQ服务来模拟DownStream下游服务，去指向一个192.168.65.193服务器上搭建的RabbitMQ服务，搭建一个联邦交换机Federation Exchange。

首先要在下游RabbitMQ中声明一个交换机和交换队列，用来接收远端的数据。这里我们直接用客户端API来快速进行声明。

```
public class DownStreamConsumer {
    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("192.168.65.112");
        factory.setPort(5672);
        factory.setUsername("admin");
        factory.setPassword("admin");
        factory.setVirtualHost("/mirror");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        channel.exchangeDeclare("fed_exchange","direct");
        channel.queueDeclare("fed_queue",true,false,false,null);
        channel.queueBind("fed_queue","fed_exchange","routKey");

        Consumer myconsumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                                       AMQP.BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("========================");
                String routingKey = envelope.getRoutingKey();
                System.out.println("routingKey >" + routingKey);
                String contentType = properties.getContentType();
                System.out.println("contentType >" + contentType);
                long deliveryTag = envelope.getDeliveryTag();
                System.out.println("deliveryTag >" + deliveryTag);
                System.out.println("content:" + new String(body, "UTF-8"));
            }
        };
        channel.basicConsume("fed_queue", true, myconsumer);
    }
}
```

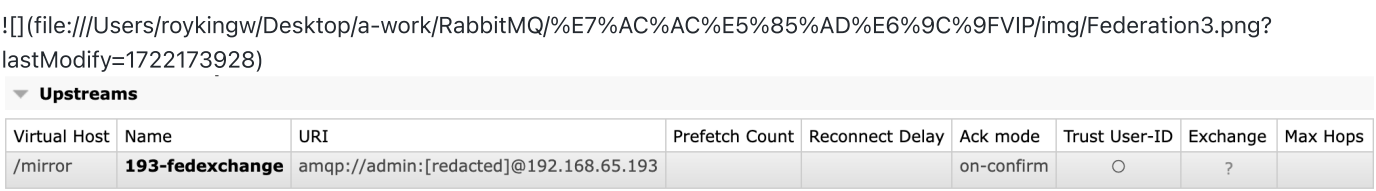
然后在下游RabbitMQ服务中配置一个上游服务。



服务的名字Name属性随意，URI指向远程服务器(配置方式参看页面上的示例)：amqp://admin:admin@192.168.65.112:5672/

> URI的详细配置方式见页面下方的示例。只不过需要注意下，DownStream和UpStream建议使用相同的虚拟机。

下面的Federated exchanges parameters和Federated queues parameters分别指定Upstream(也就是远程服务器)的Exchange和Queue。如果不指定，就是用和DownStream中相同的Exchange和Queue。如果UpStream里没有，就创建新的Exchange和Queue。



> 注意： 1、其他的相关参数，可以在页面上查看帮助。
>
> 2、关于Virtual Host虚拟机配置，如果在配置Upstream时指定了Virtual Host属性，那么在URI中就不能再添加Virtual Host配置了，默认会在Upstream上使用相同的Virtual Host。

3、配置Federation策略

接下来需要配置一个指向上游服务的Federation策略。在配置策略时可以选择是针对Exchange交换机还是针对Queue队列。配置策略时，同样有很多参数可以选择配置。最简化的一个配置如下：

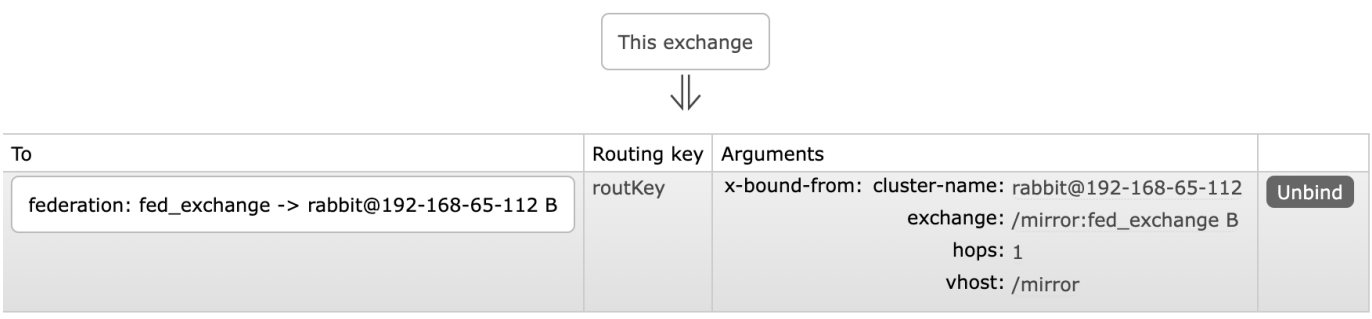
> 注意：每个策略的Definition部分，至少需要指定一个Federation目标。federation-upstream-set参数表示是以set集合的方式针对多个Upstream生效，all表示是全部Upstream。而federation-upstream参数表示只对某一个Upstream生效。

4、测试

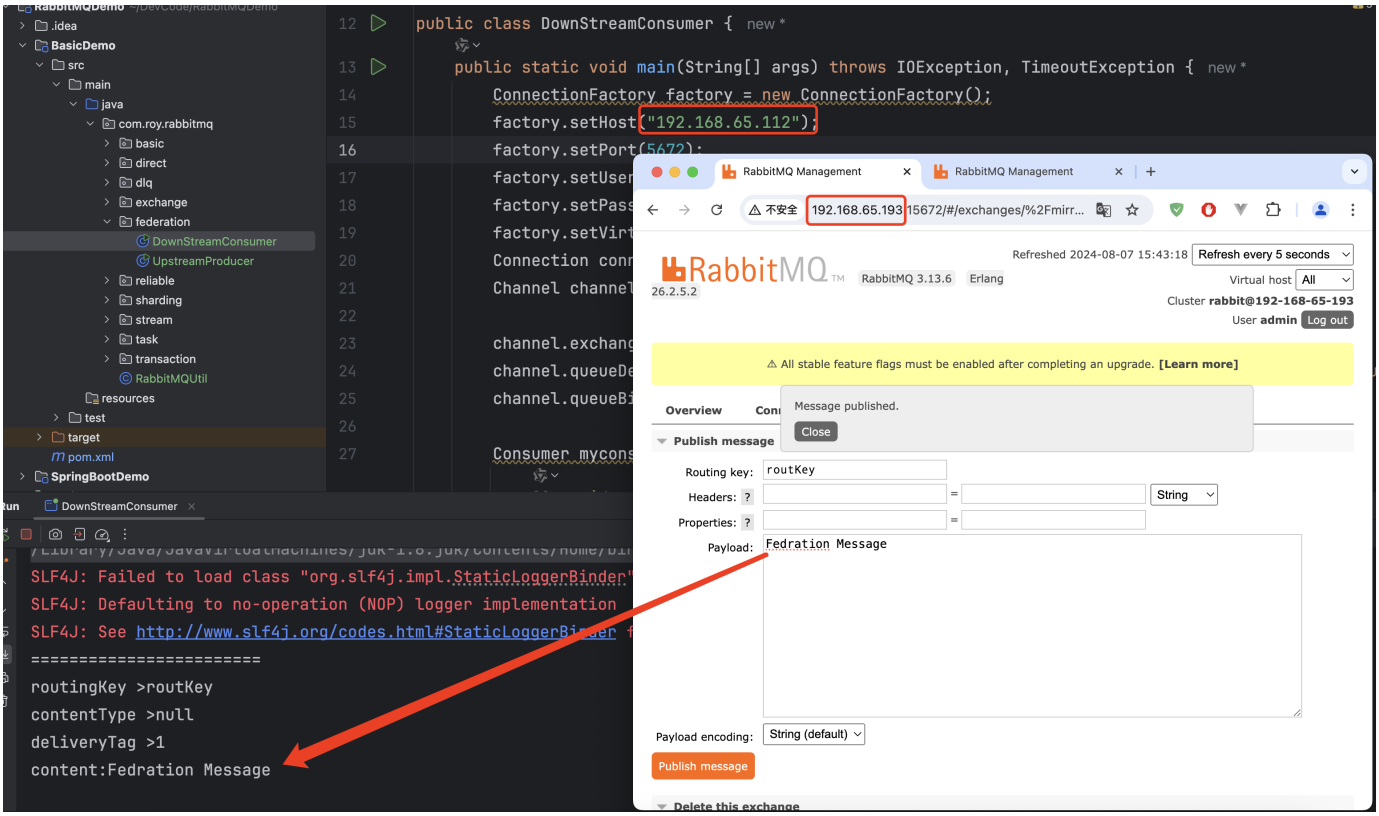
配置完Upstream和对应的策略后，进入Federation Status菜单就能看到Federation插件的执行情况。状态为running表示启动成功，如果配置出错，则会提示失败原因 这个提示非常非常简单

然后，在远程服务193的RabbitMQ服务中，可以看到对应生成的Federation交换机。

并且在fed_exchange的详情页中也能够看到绑定关系。这里要注意一下他给出了一个默认的Routing_key。



接下来就可以尝试在上游服务193的fed_exchange中发送消息，消息会同步到Local本地的联邦交换机中，从而被对应的消费者消费到。



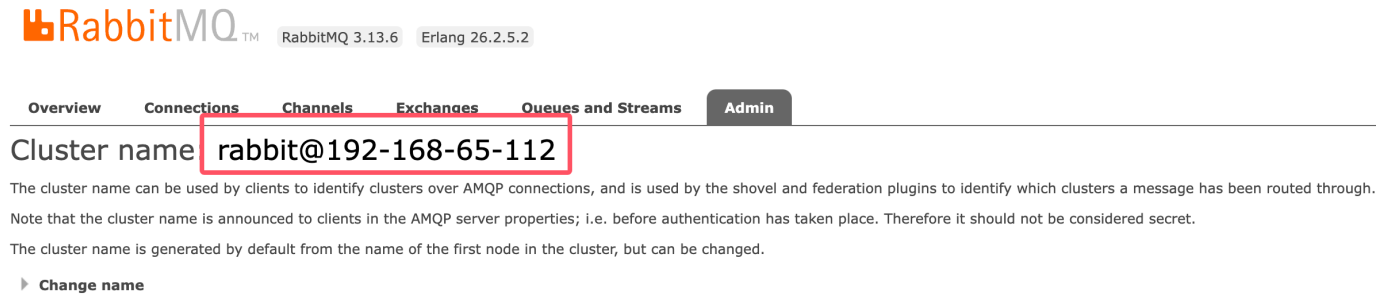
四、RabbitMQ服务高可用机制

1、RabbitMQ的集群机制

在之前几个章节，我们主要是在给大家介绍RabbitMQ的功能。这些功能，用单机环境搭建起来的RabbitMQ服务就足够体验了。但是，在企业中真实使用RabbitMQ时，单机肯定是不够的。如果只是单机RabbitMQ的服务崩溃了，那还好，大不了重启下服务就是了。但是如果是服务器的磁盘出问题了，那问题就大了。因为消息都是存储在Queue里的，Queue坏了，意味着消息就丢失了。这在生产环境上肯定是无法接受的。而RabbitMQ的设计重点就是要保护消息的安全性。

所以RabbitMQ在设计之初其实就采用了集群模式来保护消息的安全。基础的思想就是给每个Queue提供几个备份。当某一个服务的Queue坏了，至少还可以从其他Queue中获取服务。

其实对于RabbitMQ，一个节点的服务也是作为一个集群来处理的，在web控制台的admin-> cluster 中可以看到集群的名字，并且可以在页面上修改。



那么RabbitMQ是怎么考虑数据安全这回事的呢？实际上，RabbitMQ实现了两种集群模式：

- **默认的普通集群模式**：

  这种模式使用Erlang语言天生具备的集群方式搭建。这种集群模式下，集群的各个节点之间只会有相同的元数据，即队列结构，而消息不会进行冗余，只存在一个节点中。消费时，如果消费的不是存有数据的节点， RabbitMQ会临时在节点之间进行数据传输，将消息从存有数据的节点传输到消费的节点。

  很显然，这种集群模式的消息可靠性不是很高。因为如果其中有个节点服务宕机了，那这个节点上的数据就无法消费了，需要等到这个节点服务恢复后才能消费，而这时，消费者端已经消费过的消息就有可能给不了服务端正确应答，服务起来后，就会再次消费这些消息，造成这部分消息重复消费。 另外，如果消息没有做持久化，重启就消息就会丢失。

  并且，这种集群模式也不支持高可用，即当某一个节点服务挂了后，需要手动重启服务，才能保证这一部分消息能正常消费。

  所以这种集群模式只适合一些对消息安全性不是很高的场景。而在使用这种模式时，消费者应该尽量的连接上每一个节点，减少消息在集群中的传输。

- **镜像模式**：

  这种模式是在普通集群模式基础上的一种增强方案，这也就是RabbitMQ的官方HA高可用方案。需要在搭建了普通集群之后再补充搭建。其本质区别在于，这种模式会在镜像节点中间主动进行消息同步，而不是在客户端拉取消息时临时同步。

  并且在集群内部有一个算法会选举产生master和slave，当一个master挂了后，也会自动选出一个来。从而给整个集群提供高可用能力。

  这种模式的消息可靠性更高，因为每个节点上都存着全量的消息。而他的弊端也是明显的，集群内部的网络带宽会被这种同步通讯大量的消耗，进而降低整个集群的性能。这种模式下，队列数量最好不要过多。

2、搭建普通集群

接下来，我们准备三台服务器(为了方便区分，分别叫做worker1,worker2,worker3)，在三台服务器上分别搭建起RabbitMQ的服务。

然后需要在三台服务器上调整域名映射：

```
vi /etc/hosts
192.168.65.193 192-168-65-193
192.168.65.112 192-168-65-112
192.168.65.170 192-168-65-170
```

接下来还需要将各个节点上的集群名字调整为 rabbit@worker1这样的形式。每个节点的集群名字对应自己的域名。

做好这些准备工作，就可以来搭建集群了。

1、需要同步集群节点中的cookie。

默认会在 /var/lib/rabbitmq/目录下生成一个.erlang.cookie。 里面有一个字符串。我们要做的就是保证集群中三个节点的这个cookie字符串一致。

我们实验中将worker1加入到worker2的RabbitMQ集群中，所以将worker2的.erlang.cookie文件分发到worker1。

> 同步文件时注意一下文件的权限，如果文件不可读，集群启动会有问题。需要将文件的所属用户调整为rabbitmq。指令为 chown rabbitmq:rabbitmq .erlang.cookie。 另外需要注意文件的权限，必须只有当前用户可读。建议使用指令重新调整下文件权限。chmod 400 .erlang.cookie。

2、将worker2的服务加入到worker1的集群中。

首先需要保证worker1上的rabbitmq服务是正常启动的。 然后执行以下指令：

```
[root@worker2 rabbitmq]# rabbitmqctl stop_app
Stopping rabbit application on node rabbit@192-168-65-193 ...
[root@worker2 rabbitmq]# rabbitmqctl join_cluster --ram rabbit@worker2
Clustering node rabbit@worker1 with rabbit@192-168-65-193
[root@worker2 rabbitmq]# rabbitmqctl start_app
Starting node rabbit@worker1 ...
```

> --ram 表示以Ram节点加入集群。RabbitMQ的集群节点分为disk和ram。disk节点会将元数据保存到硬盘当中，而ram节点只是在内存中保存元数据。
>
> 1、由于ram节点减少了很多与硬盘的交互，所以，ram节点的元数据使用性能会比较高。但是，同时，这也意味着元数据的安全性是不如disk节点的。在我们这个集群中，worker1和worker3都以ram节点的身份加入到worker2集群里，因此，是存在单点故障的。如果worker2节点服务崩溃，那么元数据就有可能丢失。在企业进行部署时，性能与安全性需要自己进行平衡。
>
> 2、这里说的元数据仅仅只包含交换机、队列等的定义，而不包含具体的消息。因此，ram节点的性能提升，仅仅体现在对元数据进行管理时，比如修改队列queue，交换机exchange，虚拟机vhosts等时，与消息的生产和消费速度无关。
>
> 3、如果一个集群中，全部都是ram节点，那么元数据就有可能丢失。这会造成集群停止之后就启动不起来了。RabbitMQ会尽量阻止创建一个全是ram节点的集群，但是并不能彻底阻止。所以，综合考虑，官方其实并不建议使用ram节点，更推荐保证集群中节点的资源投入，使用disk节点。

加入完成后，可以在worker2的Web管理界面上看到集群的节点情况：

> 也可以用后台指令查看集群状态 rabbitmqctl cluster_status

注意：我们这里只演示了两台服务器搭建集群的过程，在实际项目中，通常建议搭建奇数台服务的集群，因为这样的集群对官方推荐的Quorum对列更友好。

3、搭建镜像集群

这样就完成了普通集群的搭建。 再此基础上，可以继续搭建**镜像集群**。

通常在生产环境中，为了减少RabbitMQ集群之间的数据传输，在配置镜像策略时，会针对固定的虚拟主机virtual host来配置。

> RabbitMQ中的vritual host可以类比为MySQL中的库，针对每个虚拟主机，可以配置不同的权限、策略等。并且不同虚拟主机之间的数据是相互隔离的。

我们首先创建一个/mirror的虚拟主机，然后再添加给对应的镜像策略：

```
[root@worker2 rabbitmq]# rabbitmqctl add_vhost /mirror
Adding vhost "/mirror" ...
[root@worker2 rabbitmq]# rabbitmqctl set_policy ha-all --vhost "/mirror" "^" '{"ha-mode":"all"}'
Setting policy "ha-all" for pattern "^" to "{"ha-mode":"all"}" with priority "0" for vhost "/mirror" ...
```

同样，这些配置的策略也可以在Web控制台操作。另外也提供了HTTP API来进行这些操作。

> 这些参数需要大致了解下。其中，pattern是队列的匹配规则，`^ 表示全部匹配`，`^ ha \` 这样的配置表示以ha开头。通常就用虚拟主机来区分就够了，这个队列匹配规则就配置成全匹配。
>
> 然后几个关键的参数：
>
> HA mode: 可选值 all , exactly, nodes。生产上通常为了保证高可用，就配all
>
> ```
> - all : 队列镜像到集群中的所有节点。当新节点加入集群时，队列也会被镜像到这个节点。
> - exactly : 需要搭配一个数字类型的参数(ha-params)。队列镜像到集群中指定数量的节点。如果集群内节点数少于这个数字，则队列镜像到集群内的所有节点。如果集群内节点少于这个数，当一个包含镜像的节点停止服务后，新的镜像就不会去另外找节点进行镜像备份了。
> - nodes: 需要搭配一个字符串类型的参数。将队列镜像到指定的节点上。如果指定的队列不在集群中，不会报错。当声明队列时，如果指定的所有镜像节点都不在线，那队列会被创建在发起声明的客户端节点上。
> ```
>
> 还有其他很多参数，可以后面慢慢再了解。

通常镜像模式的集群已经足够满足大部分的生产场景了。虽然他对系统资源消耗比较高，但是在生产环境中，系统的资源都是会做预留的，所以正常的使用是没有问题的。但是在做业务集成时，还是需要注意队列数量不宜过多，并且尽量不要让RabbitMQ产生大量的消息堆积。

> 这样搭建起来的RabbitMQ已经具备了集群特性，往任何一个节点上发送消息，消息都会及时同步到各个节点中。而在实际企业部署时，往往会以RabbitMQ的镜像队列作为基础，再增加一些运维手段，进一步提高集群的安全性和实用性。
>
> 例如，增加keepalived保证每个RabbitMQ的稳定性，当某一个节点上的RabbitMQ服务崩溃时，可以及时重新启动起来。另外，也可以增加HA-proxy来做前端的负载均衡，通过HA-proxy增加一个前端转发的虚拟节点，应用可以像使用一个单点服务一样使用一个RabbitMQ集群。这些运维方案我们就不做过多介绍了，有兴趣可以自己了解下。

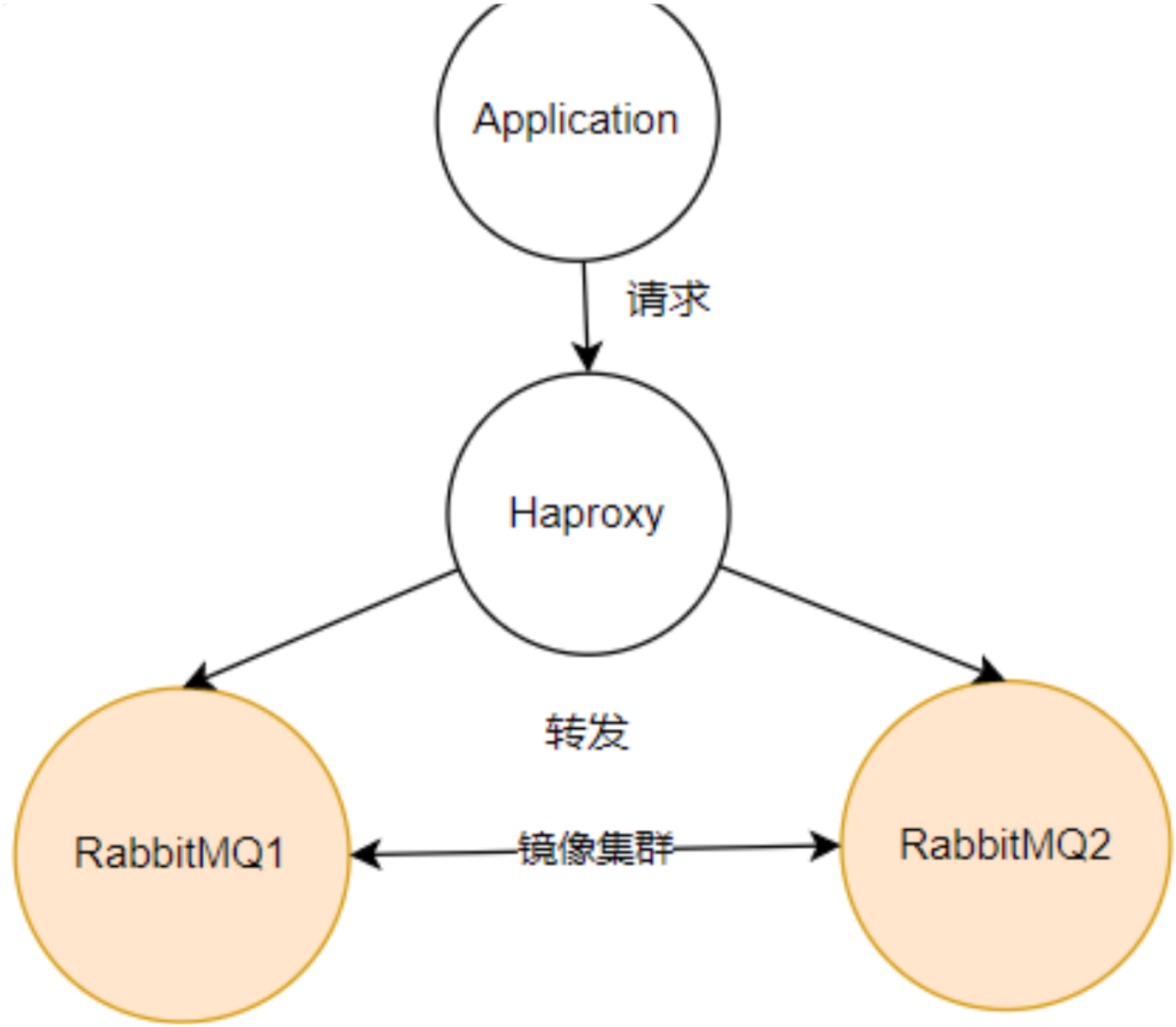
4、Haproxy+Keepalived高可用集群部署方案--了解

有了镜像集群后，应用服务只需要访问RabbitMQ的集群中任意一个服务就可以了。但是在企业应用过程中，是不是有了镜像集群就够了呢？其实高可用这事还没有这么简单。

1、Haproxy反向代理

有了镜像集群之后，客户端应用就可以访问RabbitMQ集群中任意的一个节点了。但是，不管访问哪个服务，如果这个服务崩溃了，虽然RabbitMQ集群不会丢失消息，另一个服务也可以正常使用，但是客户端还是需要主动切换访问的服务地址。

为了防止这种情况情况，可以在RabbitMQ之前部署一个Haproxy，这是一个TCP负载均衡工具。应用程序只需要访问haproxy的服务端口，Haproxy会将请求以负载均衡的方式转发到后端的RabbitMQ服务上。



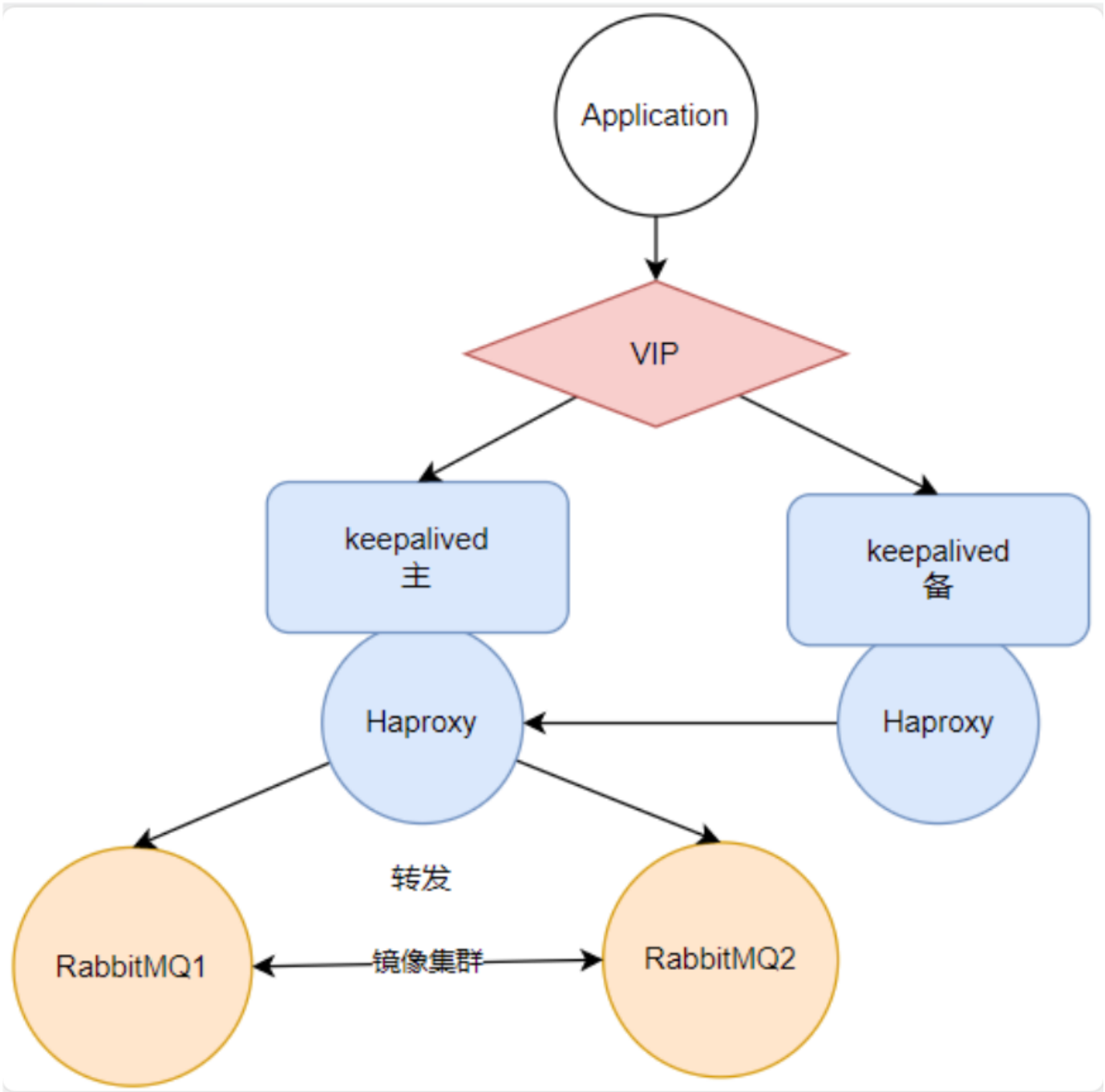
有了haproxy后，如果某一个RabbitMQ服务崩溃了，Haproxy会将请求往另外一个RabbitMQ服务转发，这样应用程序就不需要做IP切换了。此时，对于RabbitMQ来说，整个集群的服务是稳定的。

> Haproxy是一个免费开源的负载均衡工具。类似的工具还有很多，比如F5，nginx等。

2、keepalived防止haproxy单点崩溃

Haproxy保证了RabbitMQ的服务高可用，防止RabbitMQ服务单点崩溃对应用程序的影响。但是同时又带来了Haproxy的单点崩溃问题。如果Haproxy服务崩溃了，整个应用程序就完全无法访问RabbitMQ了。为了防止Haproxy单点崩溃的问题，可以引入keepalived组件来保证Haproxy的高可用。

keepalived是一个搭建高可用服务的常见工具。 他会暴露出一个虚拟IP(VIP)，并将VIP绑定到不同的网卡上。引入keepalived后，可以将VIP先绑定在已有的Haproxy服务上，然后引入一个从Haproxy作为一个备份。 当主Haproxy服务出现异常后，keepalived可以将虚拟IP转为绑定到从Haproxy服务的网卡上，这个过程称为VIP漂移。而对于应用程序，自始至终只需要访问keepalived暴露出来的VIP，感知不到VIP漂移的过程。这样就保证了Haproxy服务的高可用性。



Haproxy+Keeperalived的组合是分布式场景中经常用到的一种高可用方案。他们的部署也不麻烦，就是下载+配置+运行即可。当然，这并不是我们的重点，只要了解即可。如果你对部署操作感兴趣，想要自己搭建一下的话，可以参考下这个文章：
https://www.yuque.com/xiaochuan-5hgfq/rqiea6/xc65icrse4kkokeh 官网有对应的搭建视频。

五、课程总结

虽然MQ的功能，说起来比较简单，但是随着MQ的应用逐渐深化，所需要解决的问题也更深入。对各种细化问题的挖掘程度，很大程度上决定了开发团队能不能真正Hold得住MQ产品。通常面向互联网的应用场景，更加注重MQ的吞吐量，需要将消息尽快的保存下来，再供后端慢慢消费。而针对企业内部的应用场景，更加注重MQ的数据安全性，在复杂多变的业务场景下，每一个消息都需要有更加严格的安全保障。而在当今互联网，Kafka是第一个场景的不二代表，但是他会丢失消息的特性，让kafka的使用场景比较局限。RabbitMQ作为一个老牌产品，是第二个场景最有力的代表。当然，随着互联网应用不断成熟，也不断有其他更全能的产品冒出来，比如阿里的RocketMQ以及雅虎的Pulsar。但是不管未来MQ领域会是什么样子，RabbitMQ依然是目前企业级最为经典也最为重要的一个产品。他的功能最为全面，周边生态也非常成熟，并且RabbitMQ有庞大的Spring社区支持，本身也在吸收其他产品的各种优点，持续进化，所以未来RabbitMQ的重要性也会更加凸显。

最后，整个课程内容其实是比较多的。同时为了让这些内容能够尽量让你接触到，所以整理出了大量的资料、配置、试验。希望能够带你深入理解RabbitMQ解决各种常见问题的思路。你可能对RabbitMQ这么个年代久远的产品有过了解，但是，课上这些资料、试验你还是一定要自己从头梳理一下。因为RabbitMQ这个产品，沉淀了这么多年，积累起来的业务经验实在是太多为了。你就算用过很多年RabbitMQ，也几乎不可能触摸到RabbitMQ的全部。并且在这个过程中，你一定能找到一些你平时没有注意的技术细节。这些细节或许就是你日后解决某一个实际问题的关键。

> 【有道云笔记】四、集群实战篇.md https://note.youdao.com/s/PVZS77bp