

# 课程目标

- 1、订单延时关闭问题
- 2、死信队列和死信交换机；
- 3、延时队列的其他实现方案；
- 4、服务端流控及消费端限流；
- 5、高可用集群原理
- 6、RabbitMQ 可视化监控方案简介

## 1、订单延时关闭问题

### 1.1 业务场景

假设有一个业务场景：超过 30 分钟未付款的订单自动关闭，这个功能应该怎么实现？

思路：发一条跟订单相关的消息，30 分钟以后被消费，在消费者的代码中查询订单数据，如果支付状态是未付款，就关闭订单。

问题来了，怎么实现在指定的时候之后消息才发给消费者呢？

RabbitMQ 本身不支持延迟投递，总的来说有 2 种实现方案：

- 1、先存储到数据库，用定时任务扫描
- 2、利用 RabbitMQ 的死信队列（Dead Letter Queue）实现

定时任务比较容易实现，比如每隔 1 分钟扫描一次，查出 30 分钟之前未付款的订单，把状态改成关闭。但是如果瞬间要处理的数据量过大，比如 10 万条，把这些全部的数据查询到内存中逐条处理，也会给服务器带来很大的压力，影响正常业务的运行。

利用死信队列怎么实现呢？

这里我们要借助 RabbitMQ 消息的特性实现。

## 1.2 消息存活时间 TTL(Time To Live)

### 1.2.1 Queue 属性设置

首先，队列有一个消息过期属性。就像丰巢超过 24 小时就收费一样，通过设置这个属性，超过了指定时间的消息将会被丢弃。

这个属性叫：x-message-ttl

所有队列中的消息超过时间未被消费时，都会过期。不管是谁的包裹都一视同仁。

代码位置：com.gupaoedu.vip.mq.rabbit.springbootapi.ttl.TtlConfig.java

```
@Bean("ttlQueue")

public Queue queue() {

    Map<String, Object> map = new HashMap<String, Object>();

    map.put("x-message-ttl", 11000); // 队列中的消息未被消费 11 秒后过期

    return new Queue("GP_TTL_QUEUE", true, false, false, map);

}
```

但是这种方式似乎不是那么地灵活。所以 RabbitMQ 的消息也有单独的过期时间属性。

### 1.2.2 Message 属性设置

在发送消息的时候通过 MessageProperties 指定消息属性。

代码位置：com.gupaoedu.vip.mq.rabbit.springbootapi.ttl.TtlSender.java

```
MessageProperties messageProperties = new MessageProperties();

messageProperties.setExpiration("4000"); // 消息的过期属性，单位 ms

Message message = new Message("这条消息 4 秒后过期".getBytes(), messageProperties);
```

```
rabbitTemplate.send("GP_TTL_EXCHANGE", "gupao.ttl", message);
```

问题：如果队列 TTL 是 6 秒钟过期，msg TTL 是 10 秒钟过期，这个消息会在什么时候被丢弃？

如果同时指定了 Message TTL 和 Queue TTL，则小的那个时间生效。

有了过期时间还不够，这个消息不能直接丢弃，不然就没办法消费了。最好是丢到一个容器里面，这样就可以实现延迟消费了。

这里我们来了解一下死信的概念。

## 1.3 死信

消息过期以后，如果没有任何配置，是会直接丢弃的。我们可以通过配置让这样的消息变成死信（Dead Letter），在别的地方存储。

### 1.3.1 死信交换机 DLX 与死信队列 DLQ

队列在创建的时候可以指定一个死信交换机 DLX（Dead Letter Exchange）。死信交换机绑定的队列被称为死信队列 DLQ（Dead Letter Queue），DLX 实际上也是普通的交换机，DLQ 也是普通的队列（例如替补球员也是普通球员）。

▼ Add a new queue

Virtual host:

Name:

Durability:

Auto delete: (?)

Arguments:  =

Add	Message TTL (?)	Auto expire (?)	Max length (?)	Max length bytes (?)
	Dead letter exchange (?)	Dead letter routing key (?)	Maximum priority (?)	

也就是说，如果消息过期了，队列指定了 DLX，就会发送到 DLX。如果 DLX 绑定了 DLQ，就会路由到 DLQ。路由到 DLQ 之后，我们就可以消费了。

### 1.3.2 死信队列的实现方案

下面我们通过一个例子来演示死信队列的使用。

第一步：声明原交换机（GP\_ORI\_USE\_EXCHANGE）、原队列（GP\_ORI\_USE\_QUEUE），相互绑定。指定原队列的死信交换机（GP\_DEAD\_LETTER\_EXCHANGE）。

第二步：声明死信交换机（GP\_DEAD\_LETTER\_EXCHANGE）、死信队列（GP\_DEAD\_LETTER\_QUEUE），并且通过"#"绑定，代表无条件路由

3、最终消费者监听死信队列，在这里面实现检查订单状态逻辑。

4、生产者发送消息测试，设置消息 10 秒过期。

代码位置：com.gupaoedu.vip.mq.rabbit.javaapi.dlx.DlxConsumer

```
// 指定队列的死信交换机

Map<String, Object> arguments = new HashMap<String, Object>();
arguments.put("x-dead-letter-exchange", "GP_DEAD_LETTER_EXCHANGE");

// arguments.put("x-expires", "9000"); // 设置队列的 TTL

// arguments.put("x-max-length", 4); // 如果设置了队列的最大长度，超过长度时，先入队的消息会被发送到 DLX

// 声明队列（默认交换机 AMQP default, Direct）

// String queue, boolean durable, boolean exclusive, boolean autoDelete, Map<String, Object> arguments
channel.queueDeclare("GP_ORI_USE_QUEUE", false, false, false, arguments);

// 声明死信交换机

channel.exchangeDeclare("GP_DEAD_LETTER_EXCHANGE", "topic", false, false, false, null);

// 声明死信队列

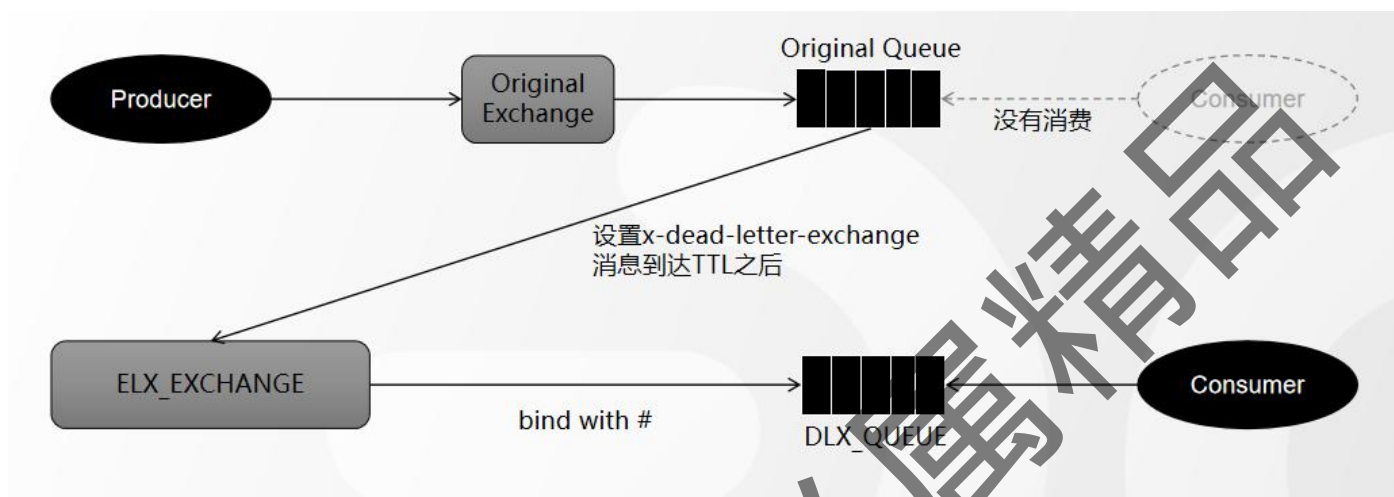
channel.queueDeclare("GP_DEAD_LETTER_QUEUE", false, false, false, null);
```

```
// 绑定，此处 Dead letter routing key 设置为 #
```

```
channel.queueBind("GP_DEAD_LETTER_QUEUE","GP_DEAD_LETTER_EXCHANGE","#");
```

```
System.out.println(" Waiting for message....");
```

### 1.3.3 死信消息流转原理



总结一下，利用消息的过期时间，过期之后投递到 DLX，路由到 DLQ，监听 DLQ，实现了延迟队列。

消息的流转流程：

生产者——原交换机——原队列（超过 TTL 之后）——死信交换机——死信队列——最终消费者

觉得这个实现巧妙的同学，把 666 刷起来。

## 1.4 延迟队列的其他实现方案

使用死信队列实现延时消息的缺点：

1) 如果统一用队列来设置消息的 TTL，当梯度非常多的情况下，比如 1 分钟，2 分钟，5 分钟，10 分钟，20 分钟，30 分钟.....需要创建很多交换机和队列来路由消息。

2) 如果单独设置消息的 TTL，则可能会造成队列中的消息阻塞——前一条消息没有出队（没有被消费），后面的消息无法投递（比如第一条消息过期 TTL 是 30min，第二条消息 TTL 是 10min。10 分钟后，即使第二条消息应该投递了，但是由于第一条

消息还未出队，所以无法投递）。

3) 可能存在一定的时间误差。

在 RabbitMQ 3.5.7 及以后的版本提供了一个插件 (rabbitmq-delayed-message-exchange) 来实现延时队列功能 (Linux 和 Windows 都可用)。同时插件依赖 Erlang/OTP 18.0 及以上。

这里下载 rabbitmq 对应的文件

<https://github.com/rabbitmq/rabbitmq-delayed-message-exchange/releases>

**v3.8.9**

This release targets RabbitMQ 3.8.9 or later versions.

### Bug Fixes

**Bindings are Now Recovered with RabbitMQ 3.8.5+ on Node Restart**

Bindings are now recovered on node boot in a way that's compatible with latest RabbitMQ releases.

GitHub issue: #149.

▼ Assets 3

rabbitmq_delayed_message_exchange-3.8.9-0199d11c.ez	49.9 KB	27 Oct 2020
Source code (zip)		26 Oct 2020
Source code (tar.gz)		26 Oct 2020

👍 1 🗨️ 4 📄 1 6 people reacted

把下载的文件 rabbitmq\_delayed\_message\_exchange-3.8.9-0199d11c.ez 放到 rabbitmq 的 plugins 下执行

```
cd /usr/lib/rabbitmq/lib/rabbitmq_server-3.8.11/plugins
```

下载插件

```
wget  
https://github.com/rabbitmq/rabbitmq-delayed-message-exchange/releases/download/3.8.9/  
rabbitmq_delayed_message_exchange-3.8.9-0199d11c.ez
```

## 安装插件

```
#启用 rabbitmq_delayed_message_exchange  
rabbitmq-plugins enable rabbitmq_delayed_message_exchange
```

## 然后重启 rabbitmq

```
service rabbitmq-server restart  
或者  
rabbitmq-server restart
```

## 5、插件使用

通过声明一个 x-delayed-message 类型的 Exchange 来使用 delayed-messaging 特性。x-delayed-message 是插件提供的类型，并不是 RabbitMQ 本身的（区别于 direct、topic、fanout、headers）。

Virtual host	Name	Type
/	(AMQP default)	direct
/	DELAY_EXCHANGE	x-delayed-message
/	amq.direct	direct
/	amq.fanout	fanout

代码位置：

```
com.gupaoedu.vip.mq.rabbit.springbootapi.dlx.delayplugin.DelayPluginConfig.  
java
```

```
@Bean("delayExchange")  
public TopicExchange exchange() {  
    Map<String, Object> argss = new HashMap<String, Object>();  
    argss.put("x-delayed-type", "direct");  
    return new TopicExchange("GP_DELAY_EXCHANGE", true, false, argss);  
}
```

生产者:

消息属性中指定 x-delay 参数。

代码位置:

com.gupaoedu.vip.mq.rabbit.springbootapi.dlx.delayplugin.DelayPluginProduce  
r.java

```
MessageProperties messageProperties = new MessageProperties();  
  
// 延迟的间隔时间，目标时刻减去当前时刻  
messageProperties.setHeader("x-delay", delayTime.getTime() - now.getTime());  
  
Message message = new Message(msg.getBytes(), messageProperties);  
  
// 不能在本地测试，必须发送消息到安装了插件的 Linux 服务端  
rabbitTemplate.send("GP_DELAY_EXCHANGE", "#", message);
```

思考：除了消息过期，还有什么情况消息会变成死信？

- 1) 消息被消费者拒绝并且未设置重回队列：(NACK || Reject) && requeue == false
- 2) 队列达到最大长度，超过了 Max length（消息数）或者 Max length bytes（字节数），最先入队的消息会被发送到 DLX。

解释：ACK: acknowledge 消息确认, NACK: Un acknowledge 没有消息确认

代码示例：com.gupaoedu.vip.mq.rabbit.javaapi.ack.AckConsumer

我们知道，RabbitMQ 的消息是存在磁盘上的，如果是内存节点，会同时存在磁盘和内存中。当 RabbitMQ 生产 MQ 消息的速度远大于消费消息的速度时，会产生大量的消息堆积，占用系统资源，导致机器的性能下降。我们想要控制服务端接收的消息的数量，应该怎么做呢？

流量控制我们可以从几方面来控制，一个是服务端，一个是消费端。



## 2、服务端流控（Flow Control）

<https://www.rabbitmq.com/configure.html>

<https://www.rabbitmq.com/flow-control.html>

<https://www.rabbitmq.com/memory.html>

<https://www.rabbitmq.com/disk-alarms.html>

### 2.1 队列长度

队列有两个控制长度的属性：

x-max-length：队列中最大存储最大消息数，超过这个数量，队头的消息会被丢弃。

x-max-length-bytes：队列中存储的最大消息容量（单位 bytes），超过这个容量，队头的消息会被丢弃。

▼ Add a new queue

Virtual host: /

Name: \*

Durability: Durable

Auto delete: (?) No

Arguments: =

Add Message TTL (?) | Auto expire (?) | **Max length (?) | Max length bytes (?)** | Dead letter exchange (?) | Dead letter routing key (?) | Maximum priority (?)

需要注意的是，设置队列长度只在消息堆积的情况下有意义，而且会删除先入队的消息，不能真正地实现服务端限流。

有没有其他办法实现服务端限流呢？

### 2.2 内存控制

我们知道，RabbitMQ 会在内存超过阈值时告警，默认为 40% 以上内存时，MQ 会主动抛出一个内存警告并阻塞所有连接（Connections）。

可以通过调整阈值来控制消息的流量。

```
rabbitmqctl set_vm_memory_high_watermark 0.6
```

如果设置成 0，则所有的消息都不能发布。

## 2.3 磁盘控制

另一种方式是通过磁盘来控制消息的发布。当磁盘剩余可用空间低于指定的值时（默认 50MB），触发流控措施。

例如：指定为磁盘的 30%或者 2GB：

<https://www.rabbitmq.com/configure.html>

```
disk_free_limit.relative = 3.0
```

```
disk_free_limit.absolute = 2GB
```

还有一种情况，虽然 Broker 消息存储得过来，但是在 push 模型下（consume，有消息就消费），消费者消费不过来了，这个时候也要对流量进行控制。

## 3、消费端限流

<https://www.rabbitmq.com/consumer-prefetch.html>

默认情况下，如果不进行配置，RabbitMQ 会尽可能快速地把队列中的消息发送到消费者。因为消费者会在本地缓存消息，如果消息数量过多，可能会导致 OOM 或者影响其他进程的正常运行。

在消费者处理消息的能力有限，例如消费者数量太少，或者单条消息的处理时间过长的情况下，如果我们希望在一定数量的消息消费完之前，不再推送消息过来，就要用到消费端的流量限制措施。

可以基于 Consumer 或者 channel 设置 prefetch count 的值，含义为 Consumer 端的最大的 unacked messages 数目。当超过这个数值的消息未被确认，RabbitMQ 会停止投递新的消息给该消费者。

```
channel.basicQos(2); // 如果超过 2 条消息没有发送 ACK，当前消费者不再接受队列消息  
channel.basicConsume(QUEUE_NAME, false, consumer);
```

参考代码：com.gupaoedu.vip.mq.rabbit.javaapi.limit

启动两个消费者，其中一个 Consumer2 消费很慢，qos 设置为 2，最多一次给它发两条消息，其他的消息都被 Consumer1 接收了。这个叫能者多劳。

## 4、高可用集群实现

### 4.1 为什么要做集群？

集群主要用于实现高可用与负载均衡。

高可用：如果集群中的某些 MQ 服务器不可用，客户端还可以连接到其他 MQ 服务器。不至于影响业务。

负载均衡：在高并发的场景下，单台 MQ 服务器能处理的消息有限，可以分发给多台 MQ 服务器。减少消息延迟。

### 4.2 RabbitMQ 如何支持集群？

应用做集群，需要面对数据同步和通信的问题。因为 Erlang 天生具备分布式的特性，所以 RabbitMQ 天然支持集群，不需要通过引入 ZK 来实现数据同步。

RabbitMQ 通过 erlang.cookie（默认路径：/var/lib/rabbitmq/）来验证身份，需要在所有节点上保持一致。这个 cookie 就像暗号一样，只要喊出清楚明白刷个 666，就知道是 Tom 的粉丝。

服务的端口是 5672，UI 的端口是 15672，集群的端口是 25672。

集群通过 25672 端口两两通信，需要开放防火墙的端口。

需要注意的是，RabbitMQ 集群无法搭建在广域网上，除非使用 federation 或者

shovel 等插件（没这个必要，在同一个机房做集群）。

### 4.3 RabbitMQ 节点类型

集群有两种节点类型，一种是磁盘节点 (Disc Node)，一种是内存节点 (RAM Node)。

磁盘节点：将元数据（包括队列名字属性、交换机的类型名字属性、绑定、vhost）放在磁盘中。未指定类型的情况下，默认为磁盘节点。



服务重启之后，存在磁盘节点中的数据还是会存在，所以像我们的持久化消息、持久化队列等，都会放置硬盘节点保存。

内存节点 (ram)：就是将元数据都放在内存里，内存节点的话，只要服务重启，该节点的所有数据将会丢失。

所以在 RabbitMQ 集群里，至少有一个磁盘节点，它用来持久保存我们的元数据，如果 RabbitMQ 是单节点运行，则默认就是磁盘节点。但是为了提高性能，其实不需要所有节点都是 disc 的节点，根据需求分配即可。

如果 RabbitMQ 集群只有一个磁盘节点，然后磁盘节点挂了，会发生什么？

可以正常的投递消息和消费消息，但是不能做以下事：

- 1、不能创建队列
- 2、不能创建交换机
- 3、不能创建用户绑定关系
- 4、不能修改用户权限

所以，考虑到高可用性，推荐在集群里保持 2 个磁盘节点，这样一个挂了，另一个还可正常工作。但上述最后一点，往集群里增加或删除节点，要求 2 个磁盘节点同时在线

#加入集群时设置节点为内存节点

```
rabbitmqctl join_cluster --ram rabbit@rabbit-node1#通过命令修改节点的类型
```

```
rabbitmqctl change_cluster_node_type disc | ram
```

## 4.3 搭建集群

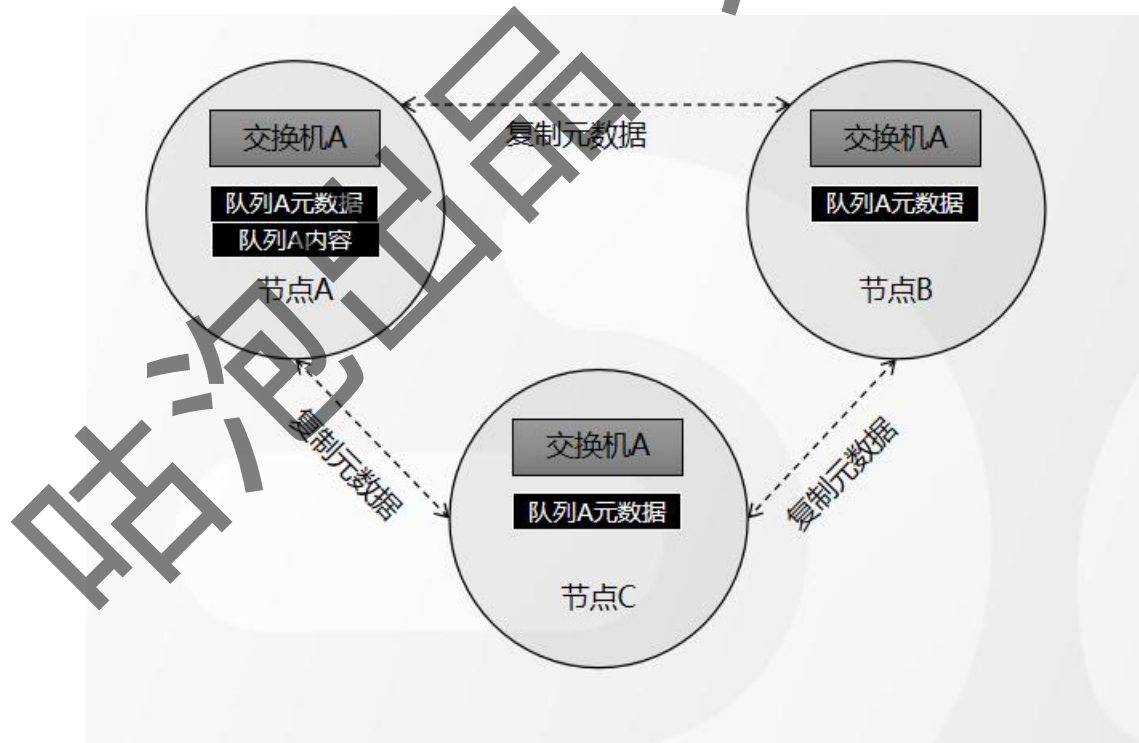
集群的配置步骤：

- 1、配置 hosts 以便相互通信
- 2、同步 erlang.cookie
- 3、加入集群 (join cluster 命令)

RabbitMQ 有两种集群模式：普通集群模式和镜像队列模式。

### 4.3.1 普通集群

**普通集群模式**下，不同的节点之间只会相互同步元数据（交换机、队列、绑定关系、Vhost 的定义），而不会同步消息。



比如，队列 A 的消息只存储在节点 A 上。节点 B 和节点 C 只同步了队列 A 的定义，但是没有同步消息。

假如生产者连接的是节点 C，要将消息通过交换机 A 路由到队列 A，最终消息还

是会转发到节点 A 上存储，因为队列 A 的内容只在节点 A 上。

同理，如果消费者连接是节点 B，要从队列 A 上拉取消息，消息会从节点 A 转发到节点 B。其它节点起到一个路由的作用，类似于指针。

这样是不是会有一个问题：如果节点 A 挂了，队列 A 的所有数据就全部丢失了。为什么不直接把消息在所有节点上复制一份？

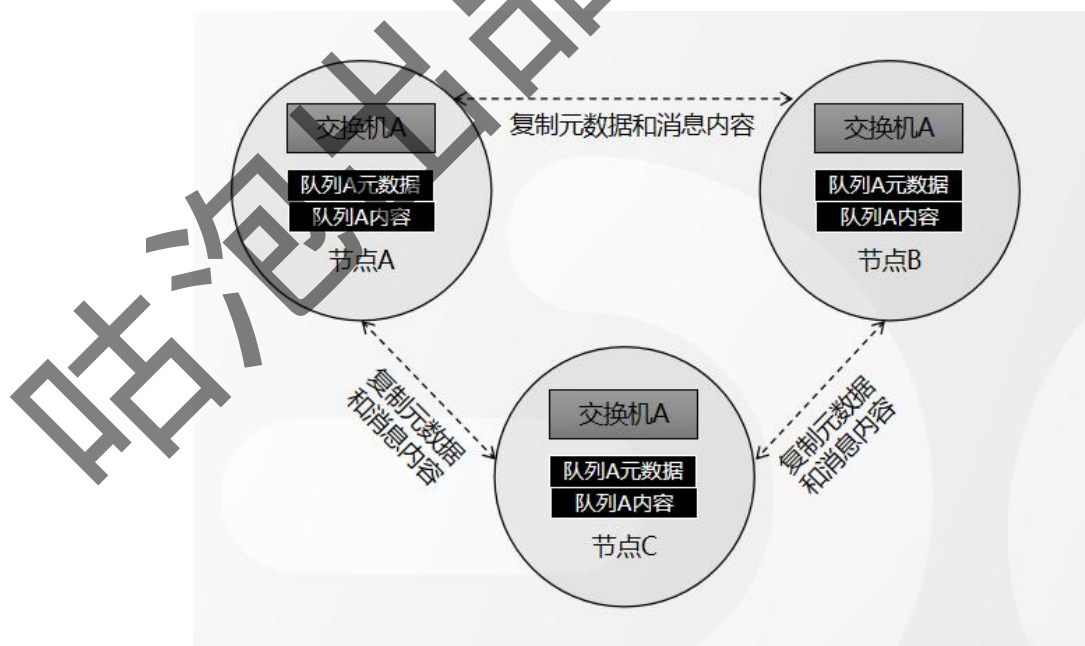
主要是出于存储和同步数据的网络开销的考虑，如果所有节点都存储相同的数据，就无法达到线性地增加性能和存储容量的目的（堆机器）。

这就是一个分片存储的思想。

当然，如果需要保证队列的高可用性，就不能用这种集群模式了，因为节点失效将导致相关队列不可用。因此我们需要第二种集群模式。

#### 4.3.2 镜像集群

第二种集群模式叫做镜像队列模式。



镜像队列模式下，消息内容会在镜像节点间同步，可用性更高。不过也有一定的副作用，系统性能会降低，节点过多的情况下同步的代价比较大。



集群模式可以通过 UI 或者 CLI 或者 HTTP 操作。

## Admin——Policies

操作方式	命令或步骤
rabbitmqctl (Windows)	<code>rabbitmqctl set_policy ha-all "^ha." '{"ha-mode":"all"}'</code>
HTTP API	<code>PUT /api/policies/%2f/ha-all {"pattern":"^ha.", "definition":{"ha-mode":"all"}}</code>
Web UI	<ol style="list-style-type: none"><li>1、avigate to Admin &gt; Policies &gt; Add / update a policy</li><li>2、Name 输入: mirror_image</li><li>3、Pattern 输入: ^ (代表匹配所有)</li><li>4、Definition 点击 HA mode, 右边输入: all</li><li>5、Add policy</li></ol>

Virtual host: 111\_vhost

Name: mirror\_image

Pattern: ^

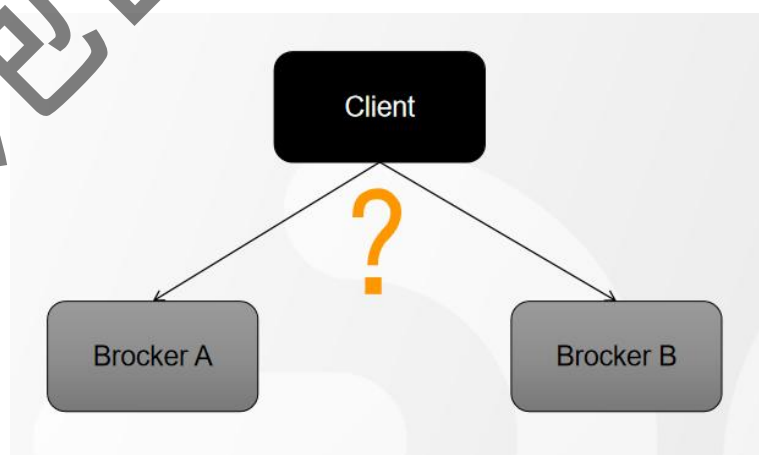
Apply to: Exchanges and queues

Priority:

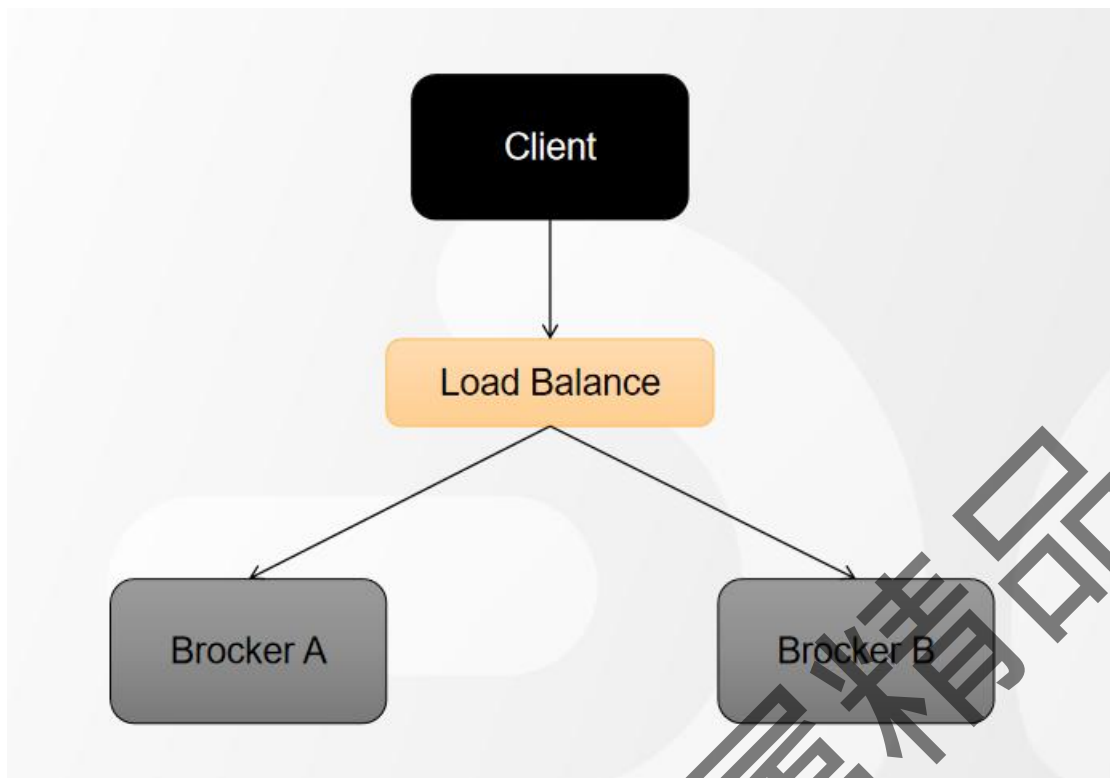
Definition: ha-mode = all

## 4.4 高可用实现原理

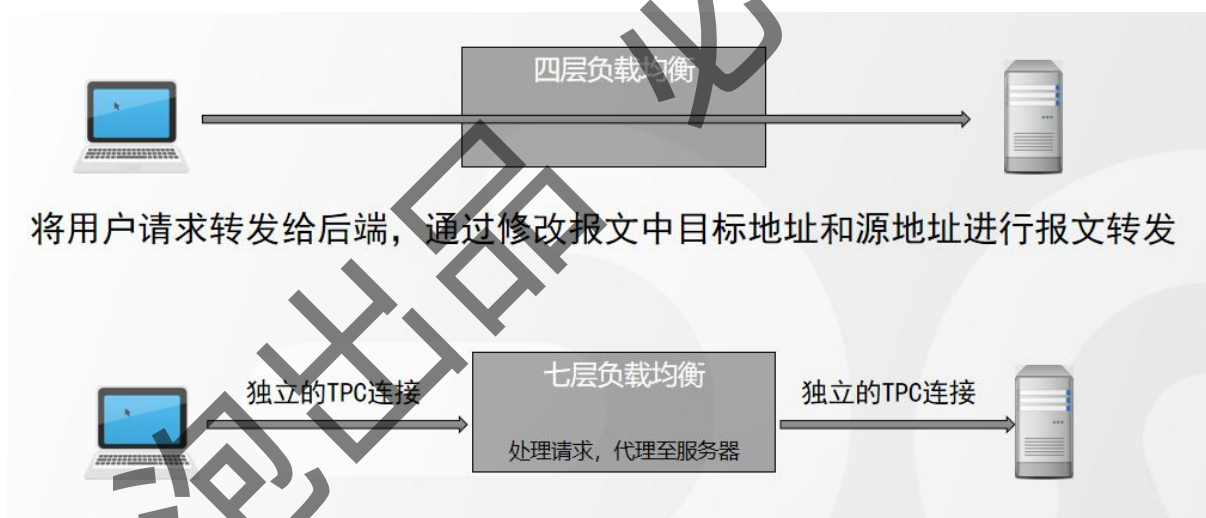
集群搭建成功后, 如果有多个内存节点, 那么生产者 and 消费者应该连接到哪个内存节点? 如果我们在客户端代码中, 需要根据一定的策略来选择要使用的服务器, 那每个地方都要修改, 客户端的代码就会出现很多的重复, 修改起来也比较麻烦。



所以需要有一个负载均衡的组件 (例如 HAProxy, LVS, Nginx), 由负载的组件来做路由。这个时候, 只需要连接到负载组件的 IP 地址就可以了。



负载分为四层负载和七层负载。



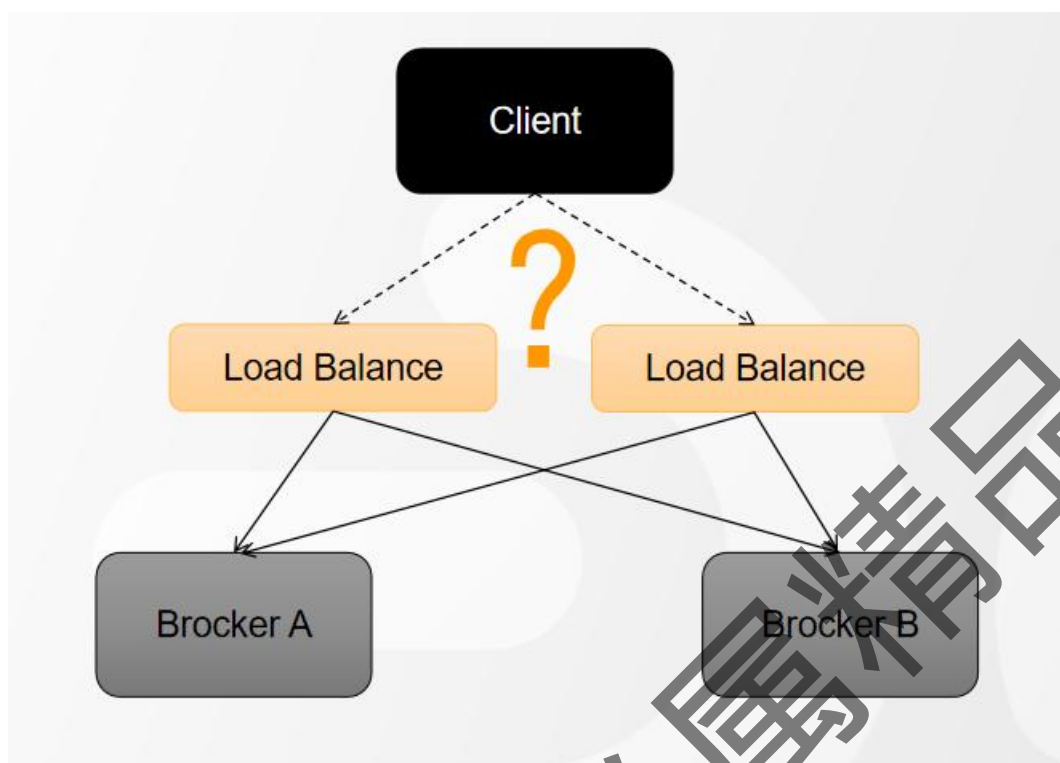
四层负载：工作在 OSI 模型的第四层，即传输层（TCP 位于第四层），它是根据 IP 端口进行转发（LVS 支持四层负载）。RabbitMQ 是 TCP 的 5672 端口。

七层负载：工作在第七层，应用层（HTTP 位于第七层）。可以根据请求资源类型分配到后端服务器（Nginx 支持七层负载；HAProxy 支持四层和七层负载）。

但是，如果这个负载的组件也挂了昵？客户端就无法连接到任意一台 MQ 的服务了。所以负载软件本身也需要做一个集群。新的问题又来了，如果有两台负载的软



件，客户端应该连哪个？



负载之上再负载？陷入死循环了。这个时候我们就要换个思路了。

我们应该需要这样一个组件：

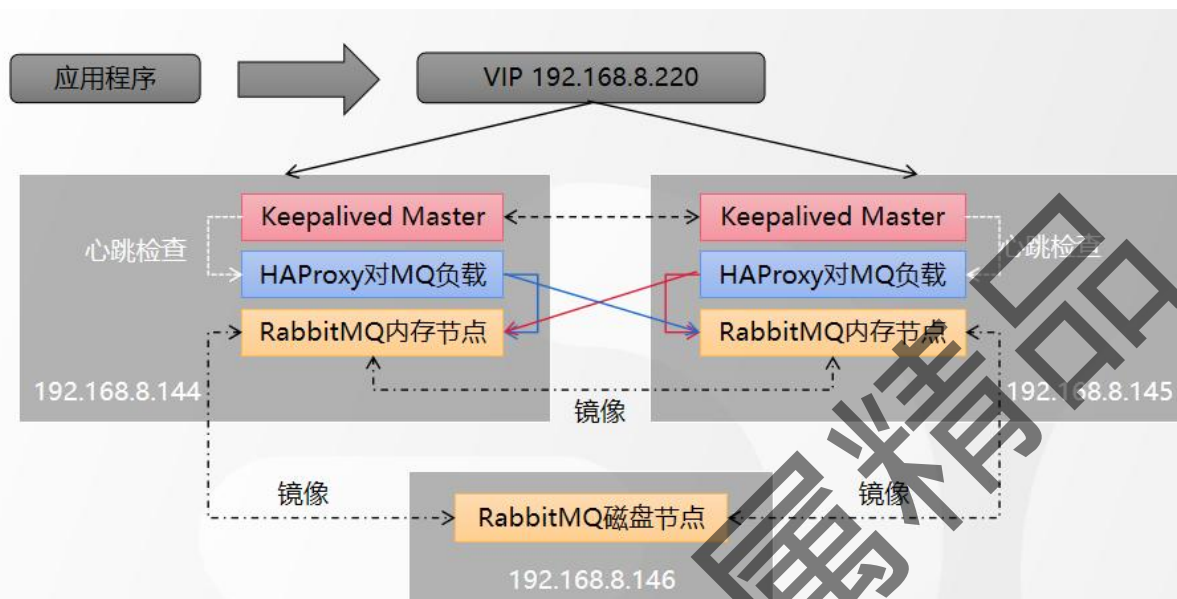
- 1、它本身有路由（负载）功能，可以监控集群中节点的状态（比如监控 HAProxy），如果某个节点出现异常或者发生故障，就把它剔除掉。
- 2、为了提高可用性，它也可以部署多个服务，但是只有一个自动选举出来的 MASTER 服务器（叫做主路由器），通过广播心跳消息实现。
- 3、MASTER 服务器对外提供一个虚拟 IP，提供各种网络功能。也就是谁抢占到 VIP，就由谁对外提供网络服务。应用端只需要连接到这一个 IP 就行了。

这个协议叫做 VRRP 协议（虚拟路由冗余协议 Virtual Router Redundancy Protocol），这个组件就是 Keepalived，它具有 Load Balance 和 High Availability 的功能。

下面我们看下用 HAProxy 和 Keepalived 如何实现 RabbitMQ 的高可用（MySQL、

Mycat、Redis 类似)。

## 4.5 基于 HAProxy+Keepalived 搭建高可用



注：HAProxy可以不跟RabbitMQ服务装在同一台机器上

规划：

内存节点 1：192.168.8.144

内存节点 2：192.168.8.145

磁盘节点：192.168.8.146

VIP：192.168.8.149

- 1、我们规划了两个内存节点，一个磁盘节点。所有的节点之间通过镜像队列的方式同步数据。内存节点用来给应用访问，磁盘节点用来持久化数据。
- 2、为了实现对两个内存节点的负载，我们安装了两个 HAProxy，监听两个 5672 和 15672 的端口。
- 3、安装两个 Keepalived，一主一备。两个 Keepalived 抢占一个 VIP 192.168.8.149。谁抢占到这个 VIP，应用就连接到谁，来执行对 MQ 的负载。

这种情况下，我们的 Keepalived 挂了一个节点，没有影响，因为 BACKUP 会

变成 MASTER，抢占 VIP。HAProxy 挂了一个节点，没有影响，我们的 VIP 会自动路由的可用的 HAProxy 服务。RabbitMQ 挂了一个节点，没有影响，因为 HAProxy 会自动负载到可用的节点。

## 5、RabbitMQ 可视化监控安装

### 5.1 Prometheus 服务端安装

以下操作皆在监控主机（192.168.56.200）上执行。

#### 2.1.1 关闭机器防火墙

```
# systemctl stop firewalld
# systemctl disable firewalld
```

```
[root@localhost ~]# systemctl stop firewalld
[root@localhost ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

#### 2.1.2 安装 go 环境

由于 Prometheus 是由 go 语言开发的，所以在安装 Prometheus 之前需要先在监控主机上安装 go 环境。这里采用源码编译的方式安装。

由于国内网络环境的原因，如果能够科学的上网，可从此地址下载最新版本的安装包：<https://golang.org/dl/>。

安装包下载以后，上传至监控主机的 /usr/local/ 目录下。

```
[root@tom7 opt]# ll
total 490164
-rw-r--r--. 1 root root 141748419 Jun  2 04:47 gol.18.3.linux-amd64.tar.gz
```

#### 2.1.3 解压安装包

```
# tar -xvf go1.18.3.linux-amd64.tar.gz
```

## 2.1.4 配置环境变量

添加/usr/local/go/bin 目录到 PATH 变量中。添加到/etc/profile 或\$HOME/.profile 都可以

```
# vim /etc/profile// 在最后一行添加

export GOROOT=/usr/local/go

export PATH=$PATH:$GOROOT/bin// wq 保存退出后 source 一下

# source /etc/profile
```

执行 go version, 如果显示版本号, 则 Go 环境安装成功。

```
[root@tom7 local]# go version
go version go1.18.3 linux/amd64
```

## 5.2 安装 Prometheus

安装包下载地址: <https://prometheus.io/download/#prometheus>

### 2.2.1 安装

将下载后安装包, 上传至 /usr/local 目录下

```
[root@tom7 opt]# ll
total 490164
-rw-r--r--. 1 root root 141748419 Jun  2 04:47 go1.18.3.linux-amd64.tar.gz
-rw-r--r--. 1 root root 59439399 Oct 16 2019 grafana-6.4.3-1.x86_64.rpm
-rw-r--r--. 1 root root 185540433 Jul  9 03:55 jdk-8u131-linux-x64.tar.gz
-rw-r--r--. 1 root root 8624587 Jul  8 21:44 node_exporter_1.2.1_linux_286.tar.gz
-rw-r--r--. 1 root root 83768501 Jul  9 04:29 prometheus-2.37.0-rc.0.linux-amd64.tar.gz
-rw-r--r--. 1 root root 51047 Jul  9 03:45 rabbitmq_delayed_message_exchange-3.8.13-0199d11c.ez
-rw-r--r--. 1 root root 6923505 Jul  8 22:41 rabbitmq_exporter_1.0.0-RC17_linux_amd64.tar.gz
-rw-r--r--. 1 root root 15816689 Jul  9 03:22 rabbitmq-server-3.8.11-1.el8.noarch.rpm
drwxr-xr-x. 2 root root 6 Oct 31 2018 rh
```

解压安装包:

```
# tar -xvf prometheus-2.37.0-rc.0.linux-amd64.tar.gz

# mv prometheus-2.37.0-rc.0.linux-amd64/ prometheus
```

### 2.2.2 启动

Prometheus 的配置文件位于 /usr/local/Prometheus/prometheus.yml , 此处采用默认配置。

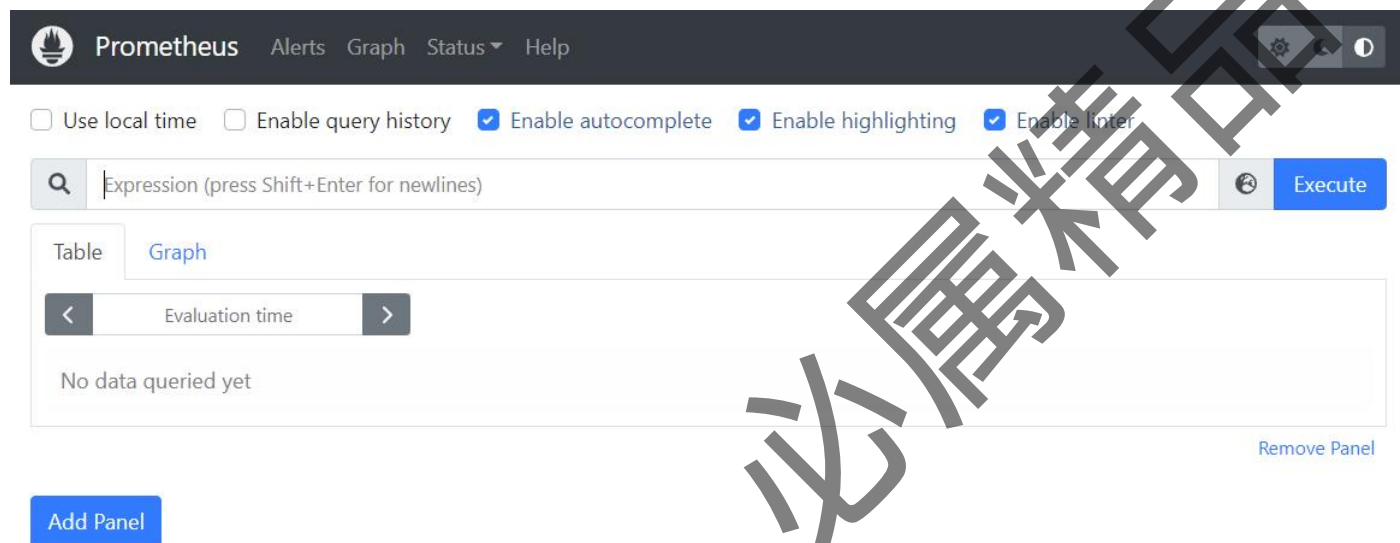
进入解压后的文件夹下，启动 Prometheus。

```
# cd prometheus

# ./prometheus --config.file=/usr/local/prometheus/prometheus.yml &
```

### 2.2.3 验证

浏览器打开 <http://192.168.8.147:9090> (IP:9090 端口) 即可打开普罗米修斯自带的监控页面



### 2.2.4 以服务的方式启动

Ctrl+C 结束掉 Prometheus 进程。创建 Prometheus 服务，让 Prometheus 以服务的方式，开机自启。

添加系统服务

```
# vim /etc/systemd/system/prometheus.service
```

将以下内容写入文件中

```
[Unit]

Description=Prometheus Monitoring System

Documentation=Prometheus Monitoring System


[Service]

ExecStart=/usr/local/prometheus/prometheus
```



```
--config.file=/usr/local/prometheus/prometheus.yml

--web.listen-address=:9090
Restart=on-failure

[Install]

WantedBy=multi-user.target
```

启动服务，设置开机自启，并检查服务开启状态

```
# systemctl daemon-reload

# systemctl enable prometheus

# systemctl start prometheus

# systemctl status prometheus
```

```
[root@localhost prometheus]# systemctl daemon-reload
[root@localhost prometheus]# systemctl enable prometheus
Created symlink from /etc/systemd/system/multi-user.target.wants/prometheus.service to /etc/systemd/system/prometheus.service.
[root@localhost prometheus]# systemctl start prometheus
[root@localhost prometheus]# systemctl status prometheus
● prometheus.service - Prometheus Monitoring System
   Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: disabled)
   Active: active (running) since 五 2019-11-01 08:49:19 CST; 15s ago
     Main PID: 1126 (prometheus)
        CGroup: /system.slice/prometheus.service
                └─1126 /usr/local/prometheus/prometheus --config.file=/usr/local/prometheus/pr...

11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49...l
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49...l
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49..."
11月 01 08:49:19 localhost.localdomain prometheus[1126]: level=info ts=2019-11-01T00:49...0
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost prometheus]#
```

## 5.3 安装 Grafana

Prometheus 自带的监控页面显示的内容没有那么直观，我们安装 grafana 来使监控数据看起来更加直观

### 2.3.1、安装 grafana

此处安装采用源码编译的方式安装。在监控主机 (192.168.8.147) /usr/local 目录下 下载安装包，并安装

```
# wget https://dl.grafana.com/oss/release/grafana-6.4.3-1.x86_64.rpm  
  
# yum localinstall grafana-6.4.3-1.x86_64.rpm
```

没有 wget 工具的，首先安装 wget 工具：

```
# yum -y install wget
```

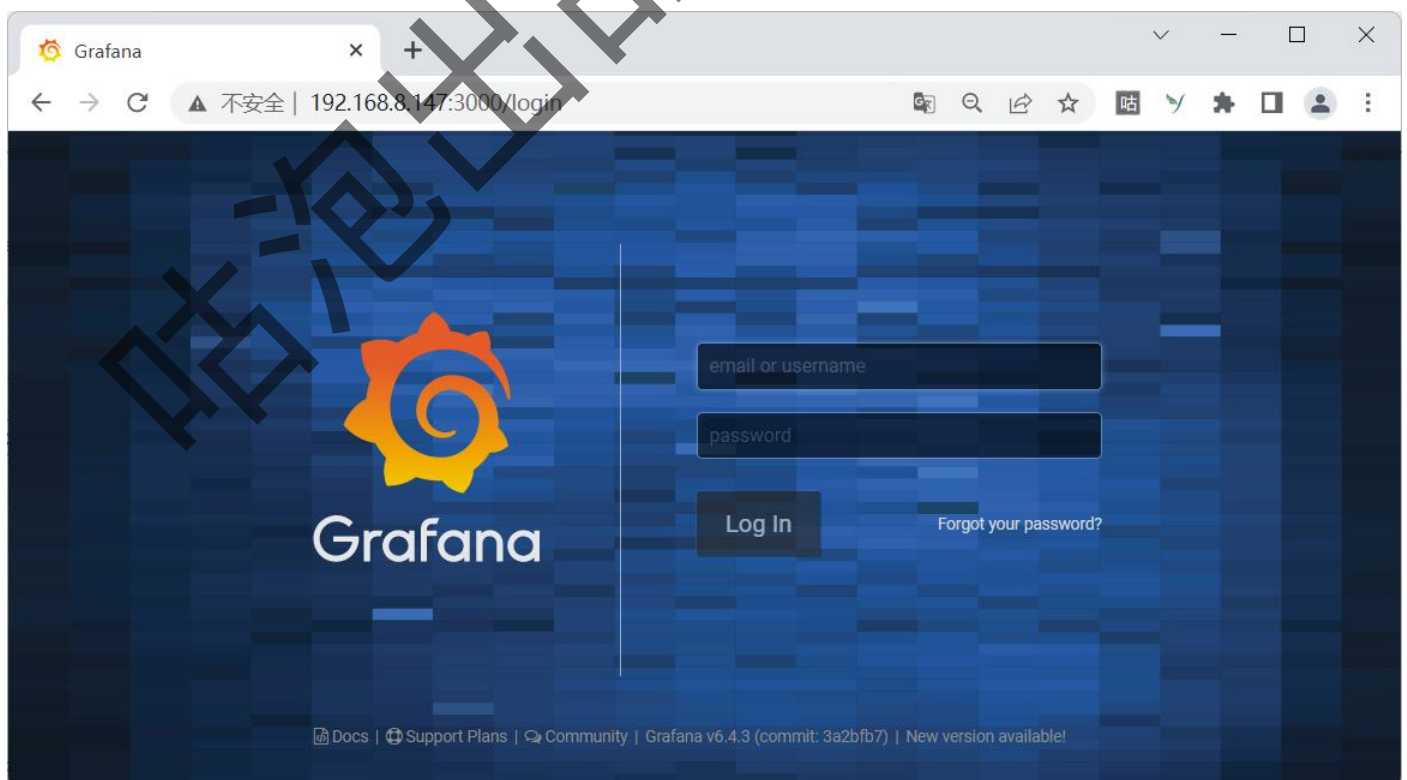
### 2.3.2、启动 grafana

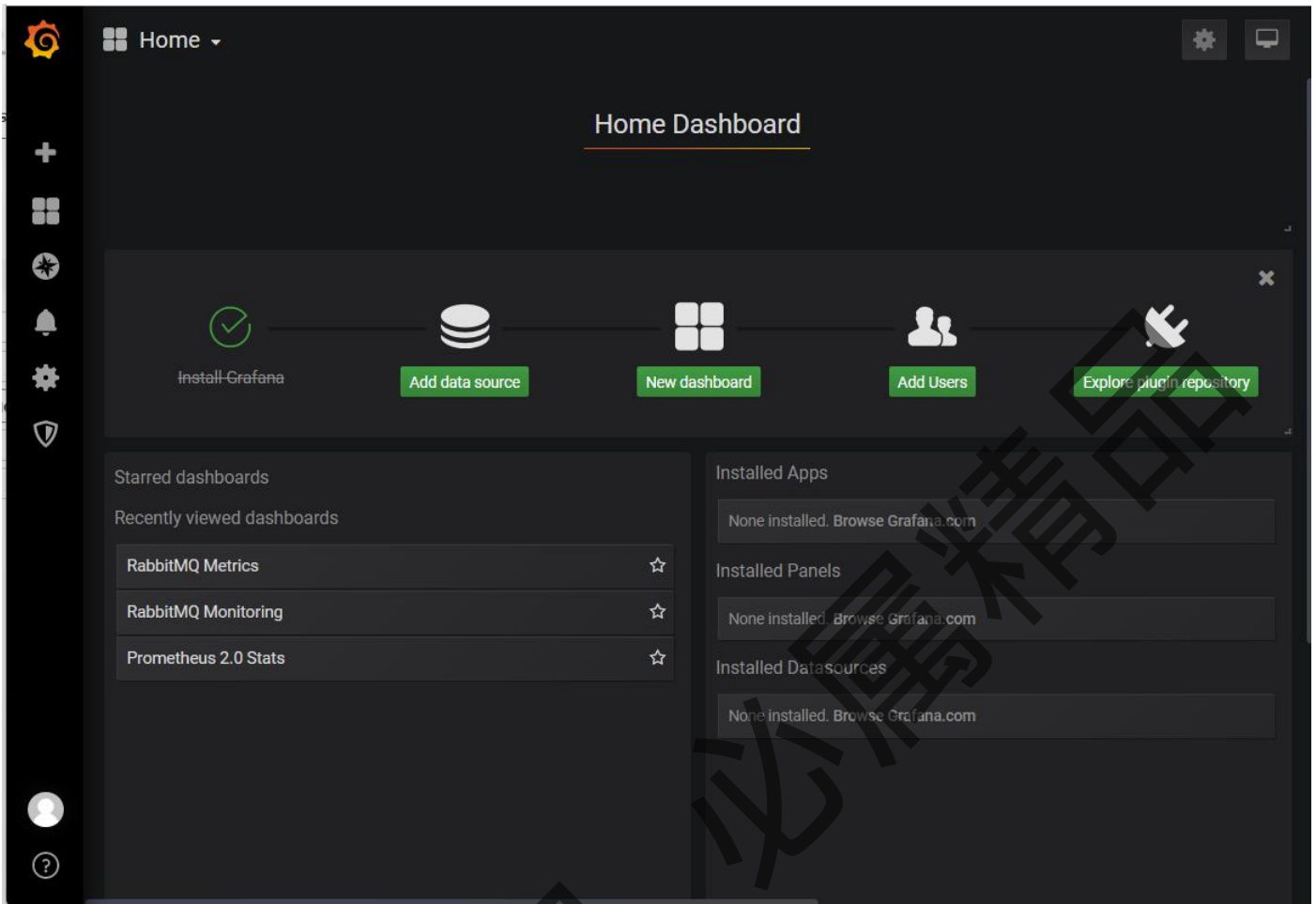
设置 grafana 服务开机自启，并启动服务

```
# systemctl daemon-reload  
  
# systemctl enable grafana-server.service  
  
# systemctl start grafana-server.service
```

### 2.3.3、访问 grafana

浏览器访问 <http://192.168.8.147:3000> (IP:3000 端口)，即可打开 grafana 页面，默认用户名密码都是 admin，初次登录会要求修改默认的登录密码

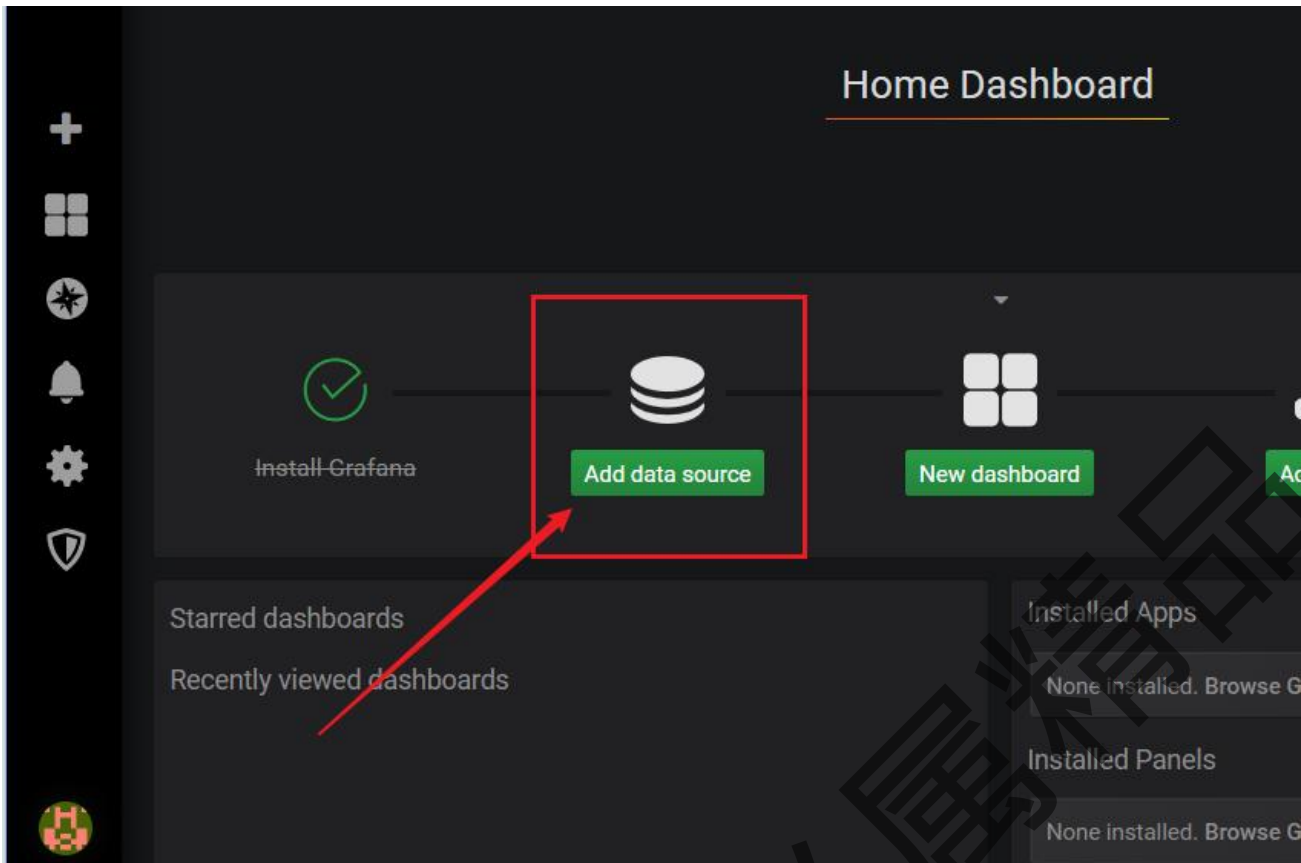




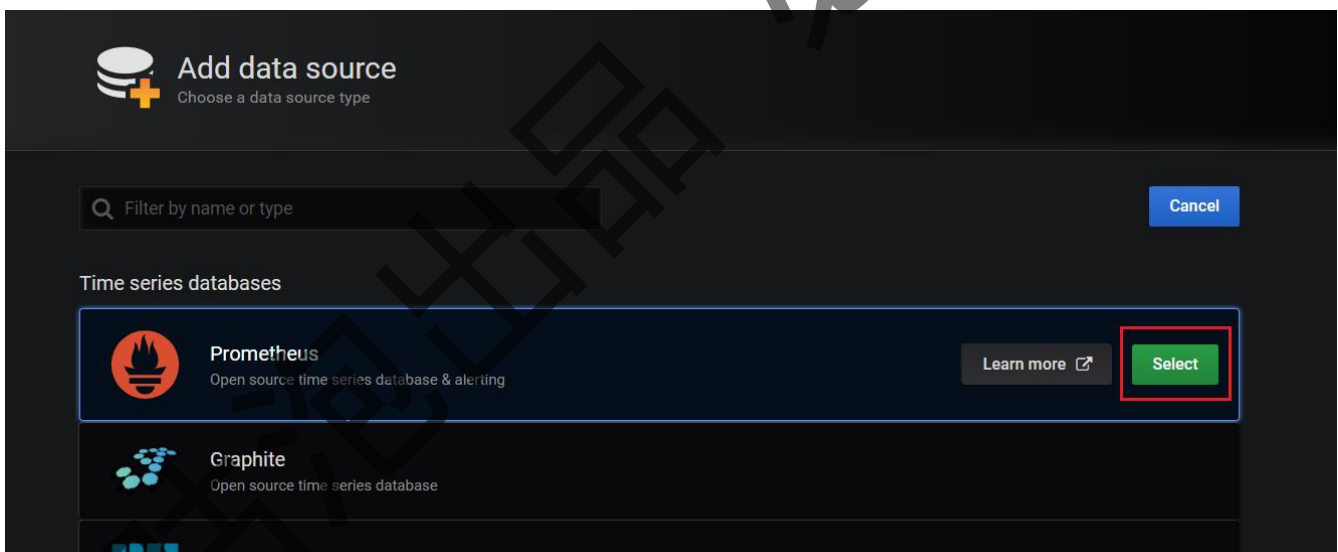
#### 2.3.4、添加 Prometheus 数据源

(1) 点击主界面的“Add data source”



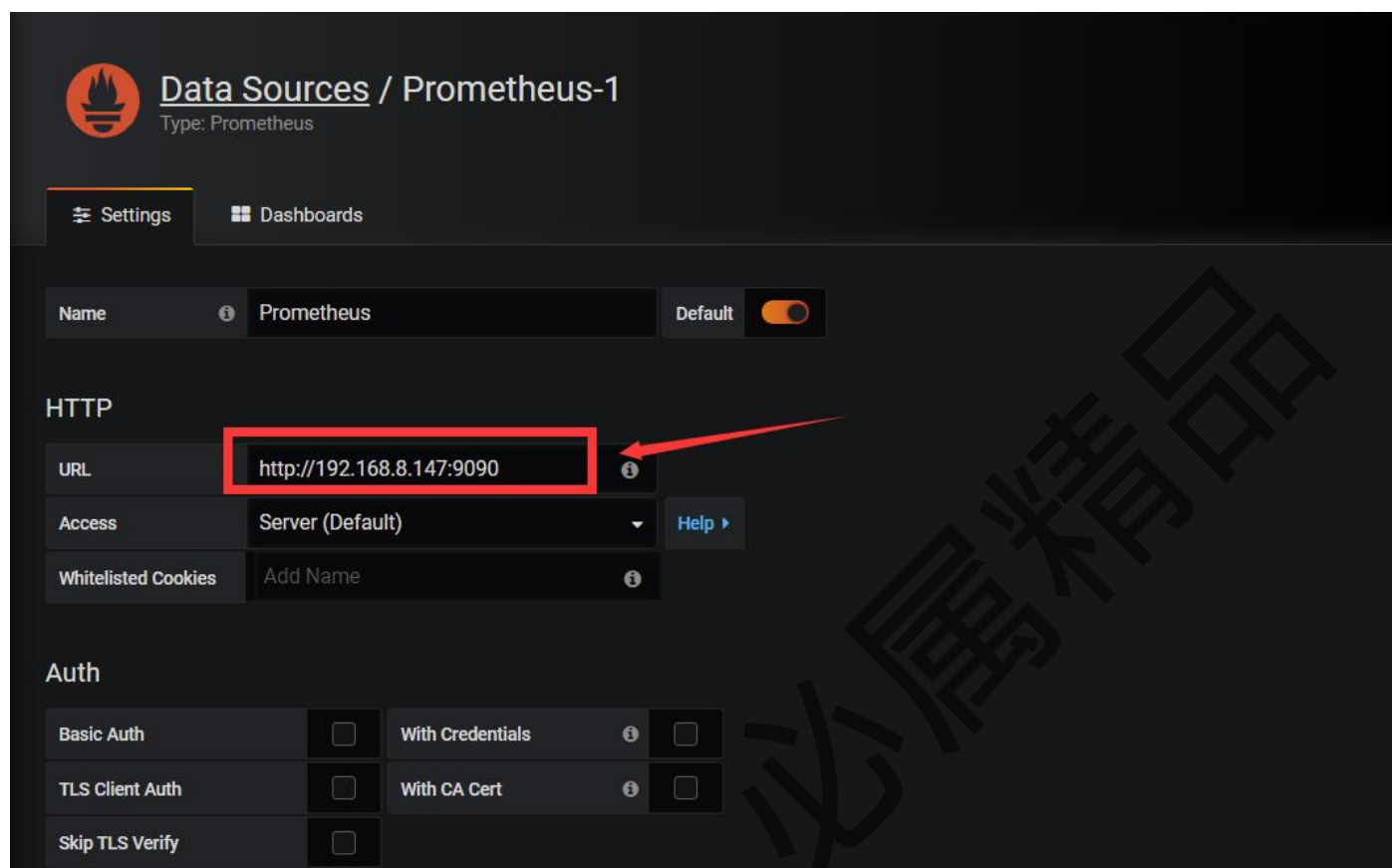


(2) 选择 Prometheus

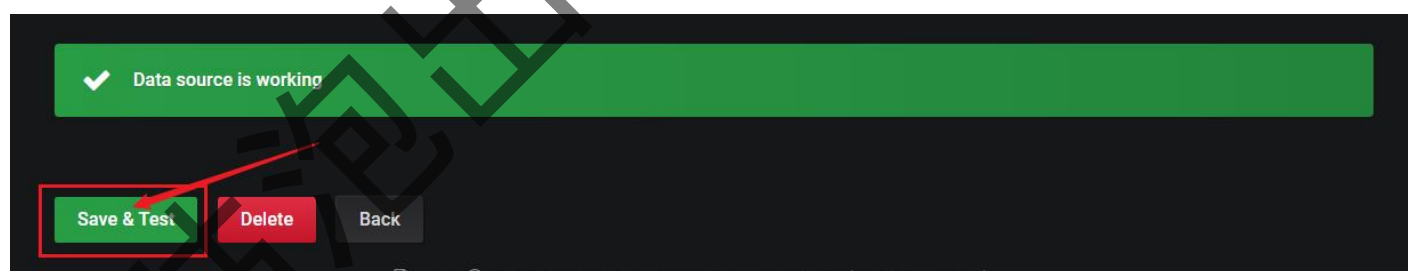


(3) 填写数据源设置项

URL 处填写 Prometheus 服务所在的 IP 地址，此处我们将 Prometheus 服务与 Grafana 安装在同一台机器上，直接填写 localhost 即可

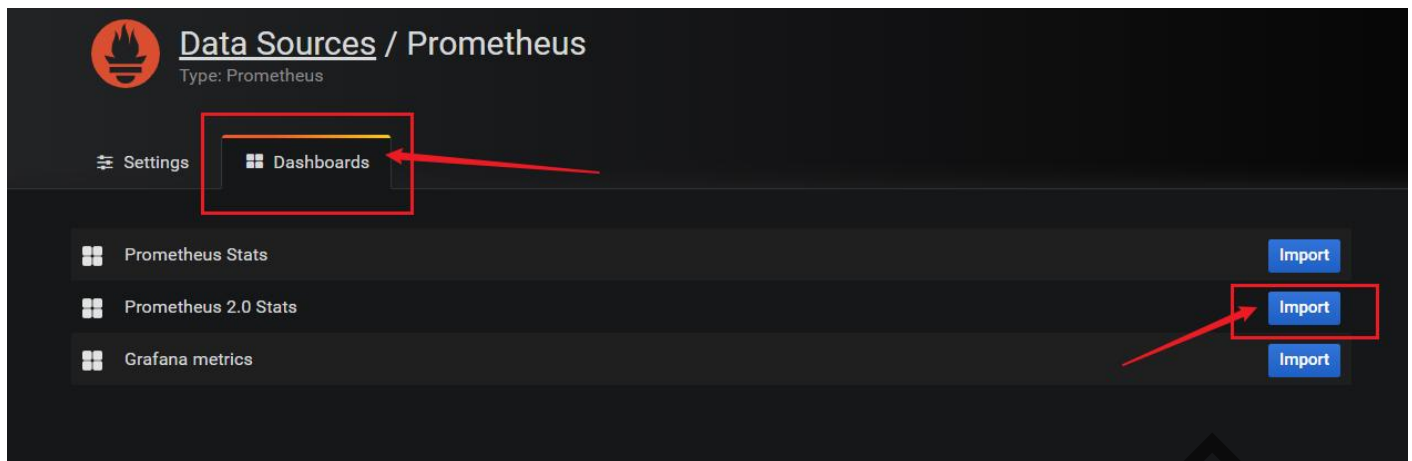


点击下方 【Save & Test】按钮，保存设置



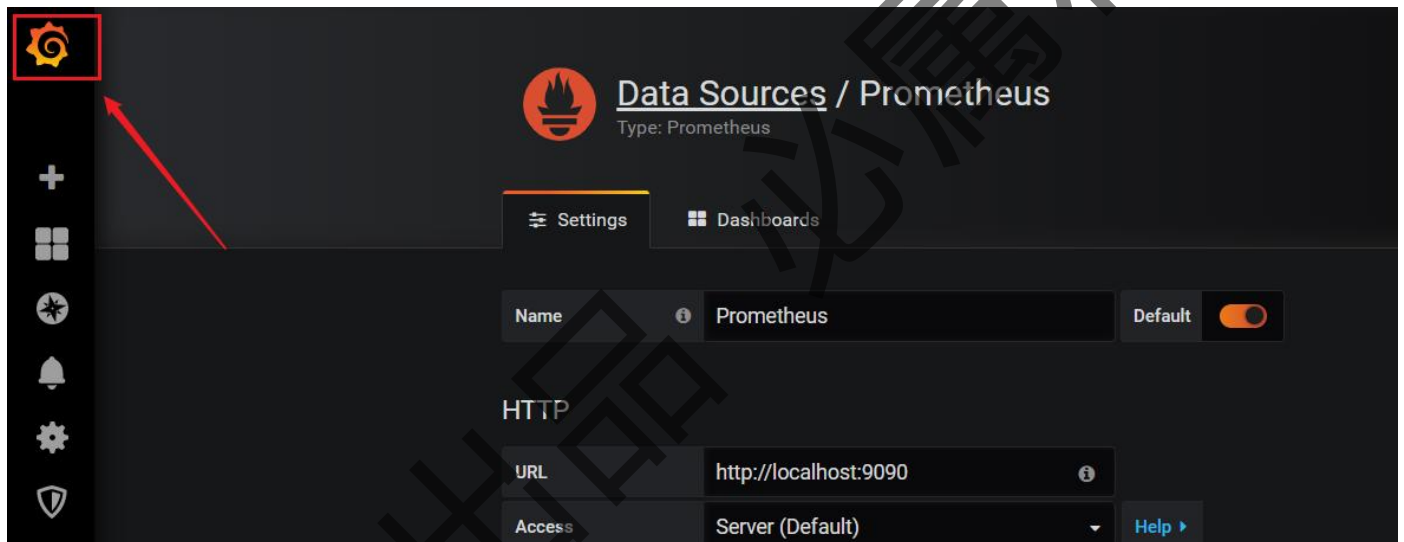
(4) Dashboards 页面选择 “Prometheus 2.0 Stats”

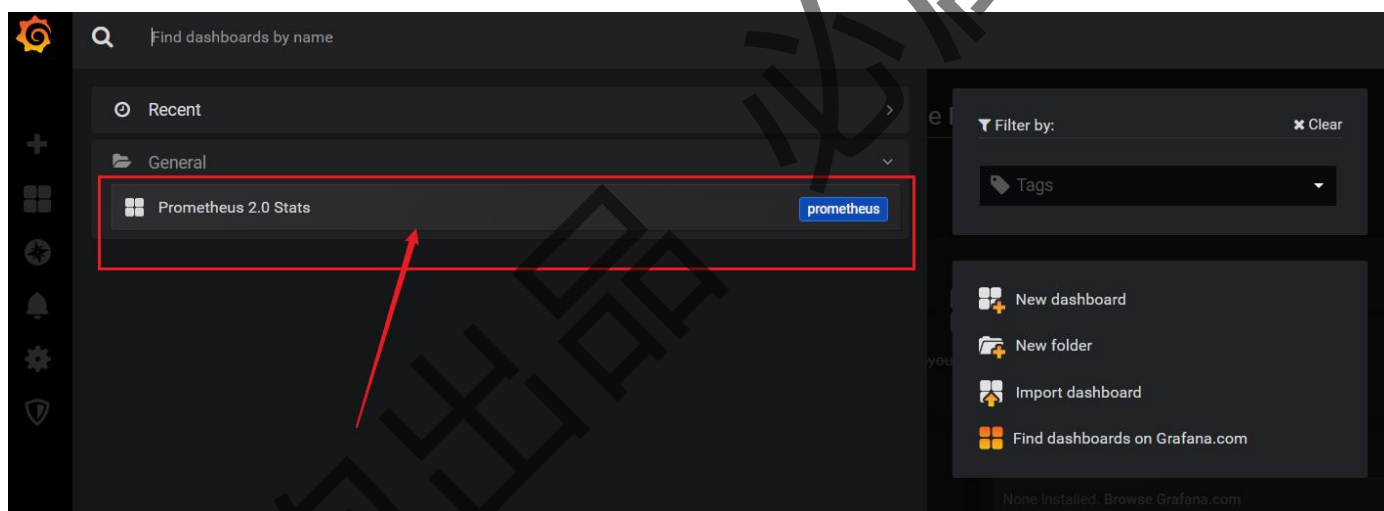
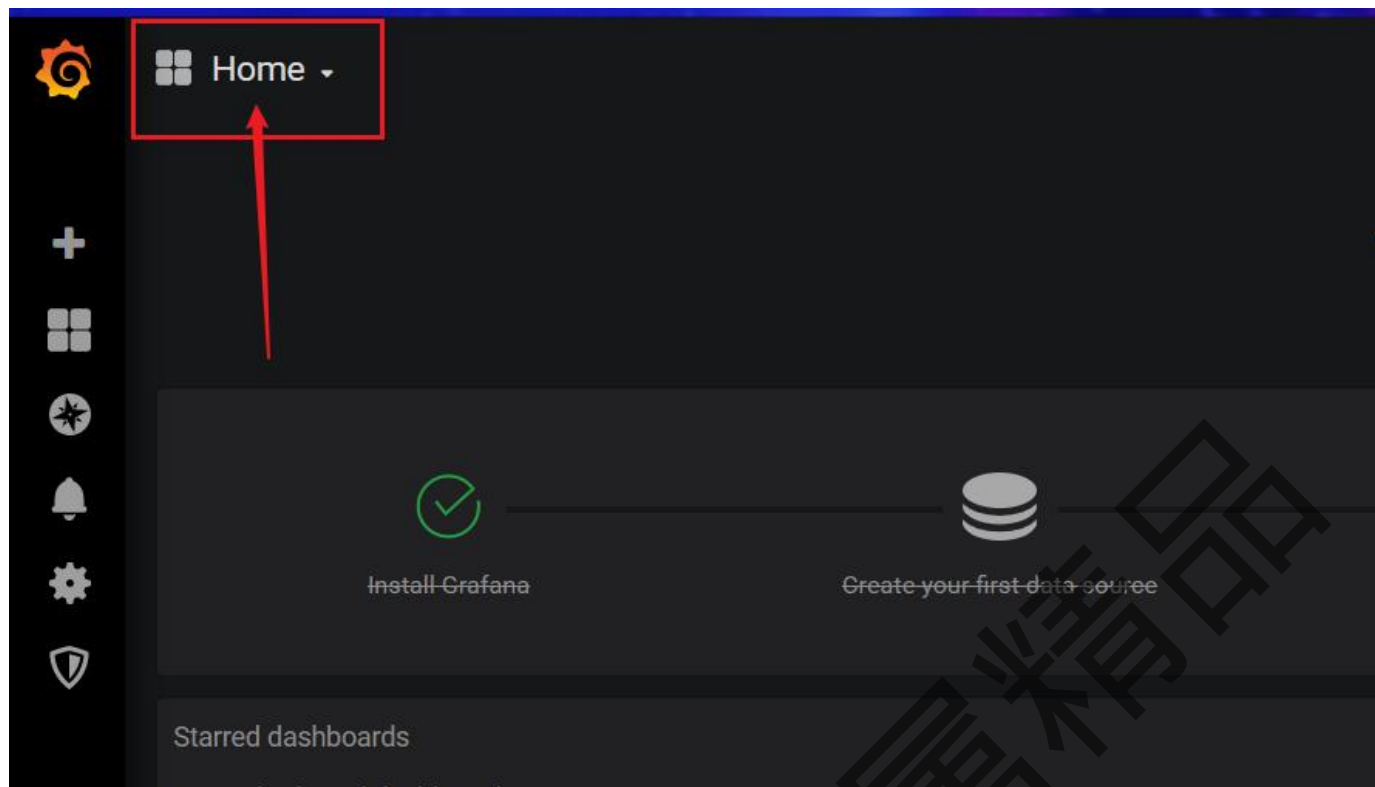
点击 Dashboards 选项卡，选择 Prometheus 2.0 Stats

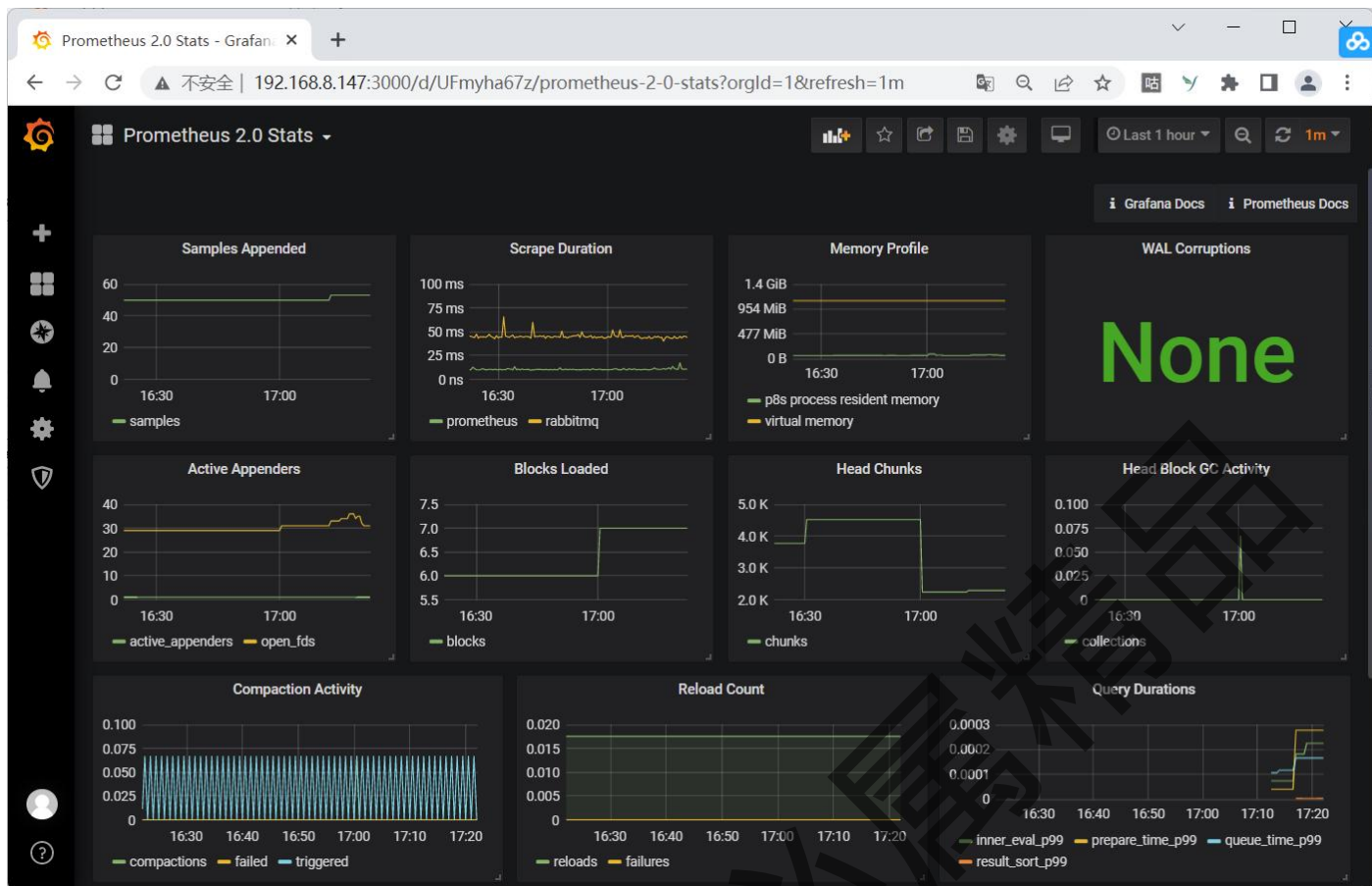


#### (5) 查看监控

点击 Grafana 图标，切换到 Grafana 主页面，然后点击 Home，选择我们刚才添加的 Prometheus 2.0 Stats，即可看到监控数据







至此 Prometheus 服务端及 Grafana 配置完成。

## 5.4 安装 rabbitmq\_exporter

以下操作皆在监控主机（192.168.8.147）上操作。

### 5.4.1、关闭机器防火墙

```
# systemctl stop firewalld
# systemctl disable firewalld
```

切换到/usr/local/

下载文件 [https://github.com/kbudde/rabbitmq\\_exporter/releases](https://github.com/kbudde/rabbitmq_exporter/releases)

解压文件

```
tar -zxf rabbitmq_exporter_1.0.0-RC17_linux_amd64.tar.gz
cd rabbitmq_exporter_1.0.0-RC17_linux_amd64
```

```
mv rabbitmq_exporter_1.0.0-RC17_linux_amd64 rabbitmq_exporter
```

## 启动监听

```
RABBIT_USER=cloudchef RABBIT_PASSWORD=【密码】 OUTPUT_FORMAT=JSON PUBLISH_PORT=9090  
RABBIT_URL=http://192.168.85.117:15672 ./rabbitmq_exporter
```

mq 的用户和密码，服务端口，ui 地址，如下

```
RABBIT_USER=admin  
  
RABBIT_PASSWORD=123456  
  
OUTPUT_FORMAT=JSON PUBLISH_PORT=9099  
  
RABBIT_URL=http://192.168.8.149:15672  
  
RABBIT_USER=admin RABBIT_PASSWORD=123456 OUTPUT_FORMAT=JSON PUBLISH_PORT=9099  
RABBIT_URL=http://192.168.8.149:15672 nohup ./rabbitmq_exporter &
```

查看端口监听是否已经生效

```
ss -nutlp | grep 9099
```

打开 prometheus 配置文件

```
vim /usr/local/prometheus/prometheus.yml
```

给 prometheus 配置最后添加以下内容

```
- job_name: 'rabbitmq'  
  static_configs:  
    - targets: ['192.168.8.149:9099']
```

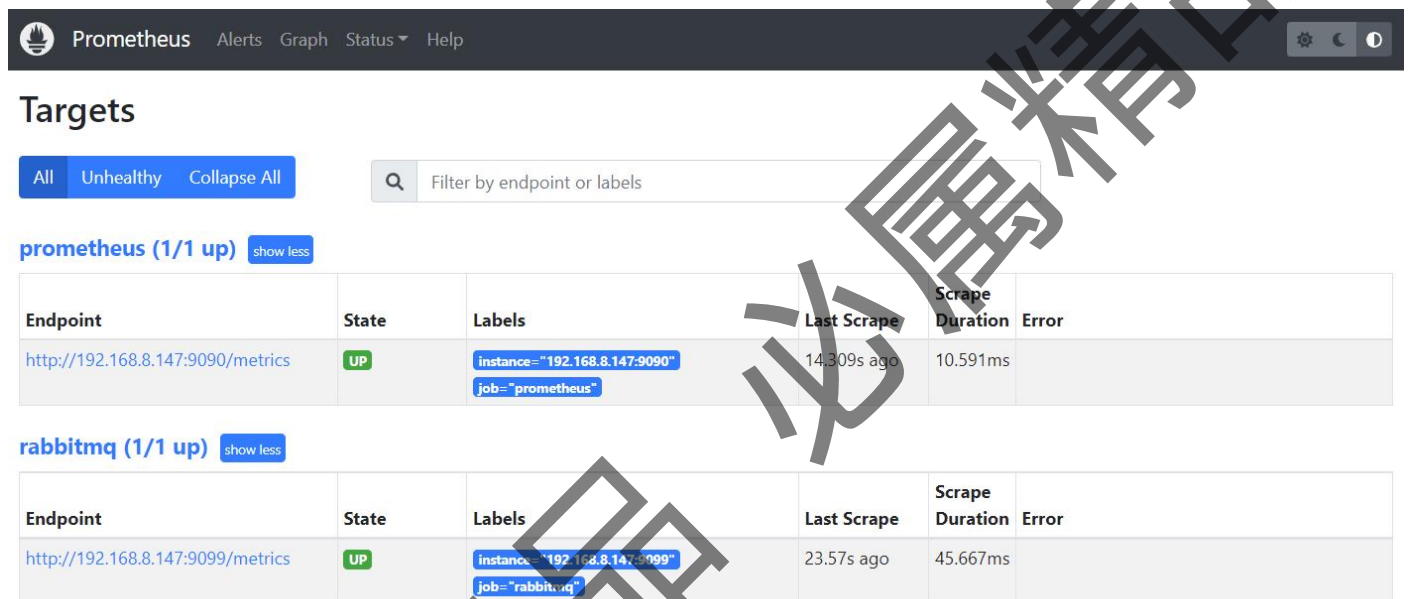
重启 prometheus

```
systemctl restart prometheus
```

如果遇到时间不同步的问题，执行以下操作

```
/usr/local/prometheus/prometheus \
--config.file="/usr/local/prometheus/prometheus.yml" \
--storage.tsdb.path="/usr/local/prometheus/tsdb_data" \
--web.enable-lifecycle \
--web.enable-admin-api &
```

浏览器访问 <http://192.168.8.147:9090/targets> 查看监控信息



The screenshot shows the Prometheus web interface at the 'Targets' page. The top navigation bar includes 'Prometheus', 'Alerts', 'Graph', 'Status', and 'Help'. Below the navigation bar, there are tabs for 'All', 'Unhealthy', and 'Collapse All', along with a search filter 'Filter by endpoint or labels'. The main content area displays two target groups: 'prometheus (1/1 up)' and 'rabbitmq (1/1 up)'. Each group has a 'show less' button. The 'prometheus' group shows a single target with endpoint 'http://192.168.8.147:9090/metrics', state 'UP', labels 'instance="192.168.8.147:9090"' and 'job="prometheus"', last scrape '14.309s ago', and scrape duration '10.591ms'. The 'rabbitmq' group shows a single target with endpoint 'http://192.168.8.147:9099/metrics', state 'UP', labels 'instance="192.168.8.147:9099"' and 'job="rabbitmq"', last scrape '23.57s ago', and scrape duration '45.667ms'.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.8.147:9090/metrics	UP	instance="192.168.8.147:9090" job="prometheus"	14.309s ago	10.591ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.8.147:9099/metrics	UP	instance="192.168.8.147:9099" job="rabbitmq"	23.57s ago	45.667ms	

可以看到，Linux 机器已经加入进来。

#### 5.4.2、配置 Grafana

添加 dashboard

Grafana 官方为我们提供了很多 dashboard 页面，可直接下载使用。浏览器访

问 <https://grafana.com/grafana/dashboards> 下载所需要的 dashboard 页面



Filter by:

Category **Rabbitmq**

Panel **All**

Data Source **All**

Collector Types **All**

Sort by **Downloads**

Share your dashboards  
Export any dashboard from

Search dashboards

37 results ✕ Clear all filters

**RabbitMQ-Overview**  
☆ No ratings  
4.23M downloads  
prometheus

**RabbitMQ-Quorum-Queues-Raft**  
☆ No ratings  
2.03M downloads  
Prometheus

**Erlang-Distribution**  
☆ No ratings  
2.00M downloads  
prometheus

**Erlang-Distributions-Compare**

**RabbitMQ-Stream**

**RabbitMQ Metrics**

**RabbitMQ Monitoring**  
☆ No ratings  
61.0 downloads  
Prometheus

**RabbitMQ Overview**  
☆ No ratings  
50.0 downloads  
Prometheus

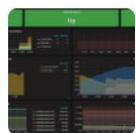
**Easy RabbitMQ (K8S)**  
★ 5/5 1 rating  
29.0 downloads  
Prometheus

选择数据源为 Prometheus，然后我们选择第一个 dashboard

复制 dashboard Id



All dashboards » [RabbitMQ Monitoring](#)



### RabbitMQ Monitoring by Sébastien Helbert

DASHBOARD

Rabbitmq stats and alerting using RabbitMQ Exporter  
([https://github.com/kbudde/rabbitmq\\_exporter](https://github.com/kbudde/rabbitmq_exporter))

Last updated: 2 years ago

Start with Grafana Cloud and the new FREE tier. Includes 10K series Prometheus or Graphite Metrics and 50gb Loki Logs

Downloads: 61

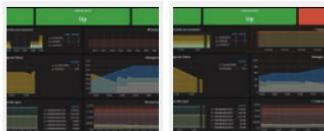
Reviews: 0

[Add your review!](#)

Overview

Revisions

Reviews



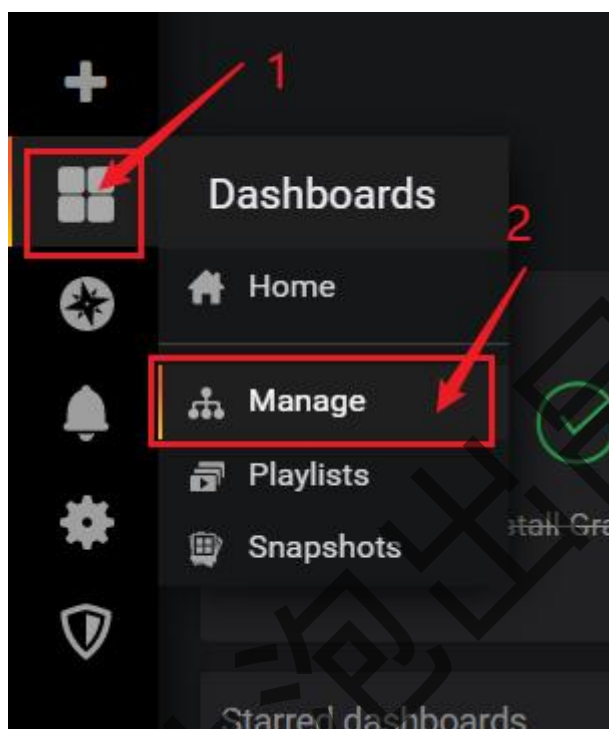
Get this dashboard:

4279

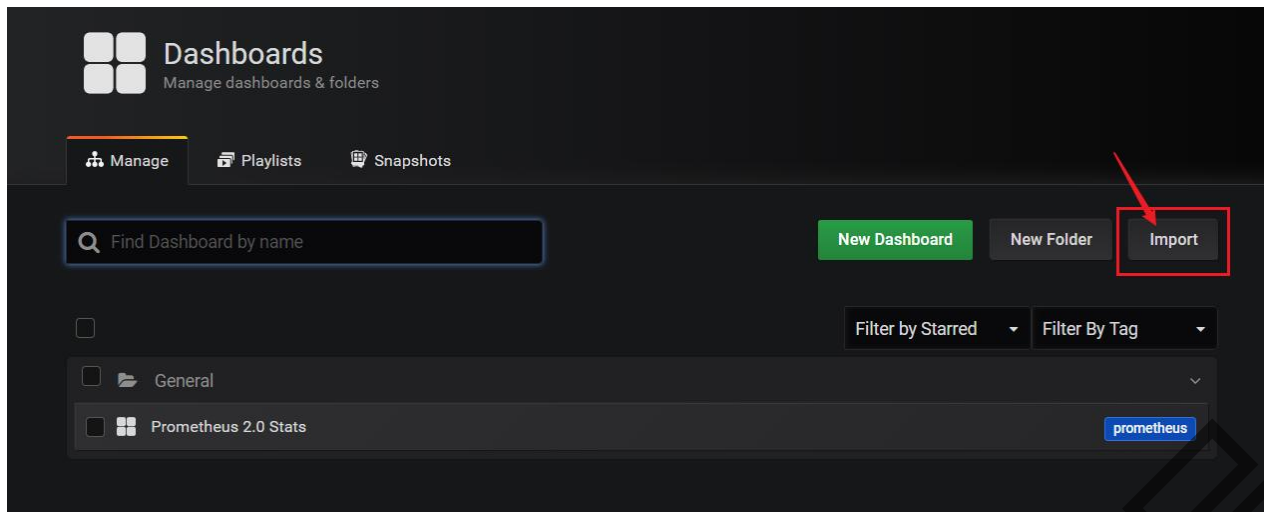
[Copy ID to Clipboard](#)

[Download JSON](#)

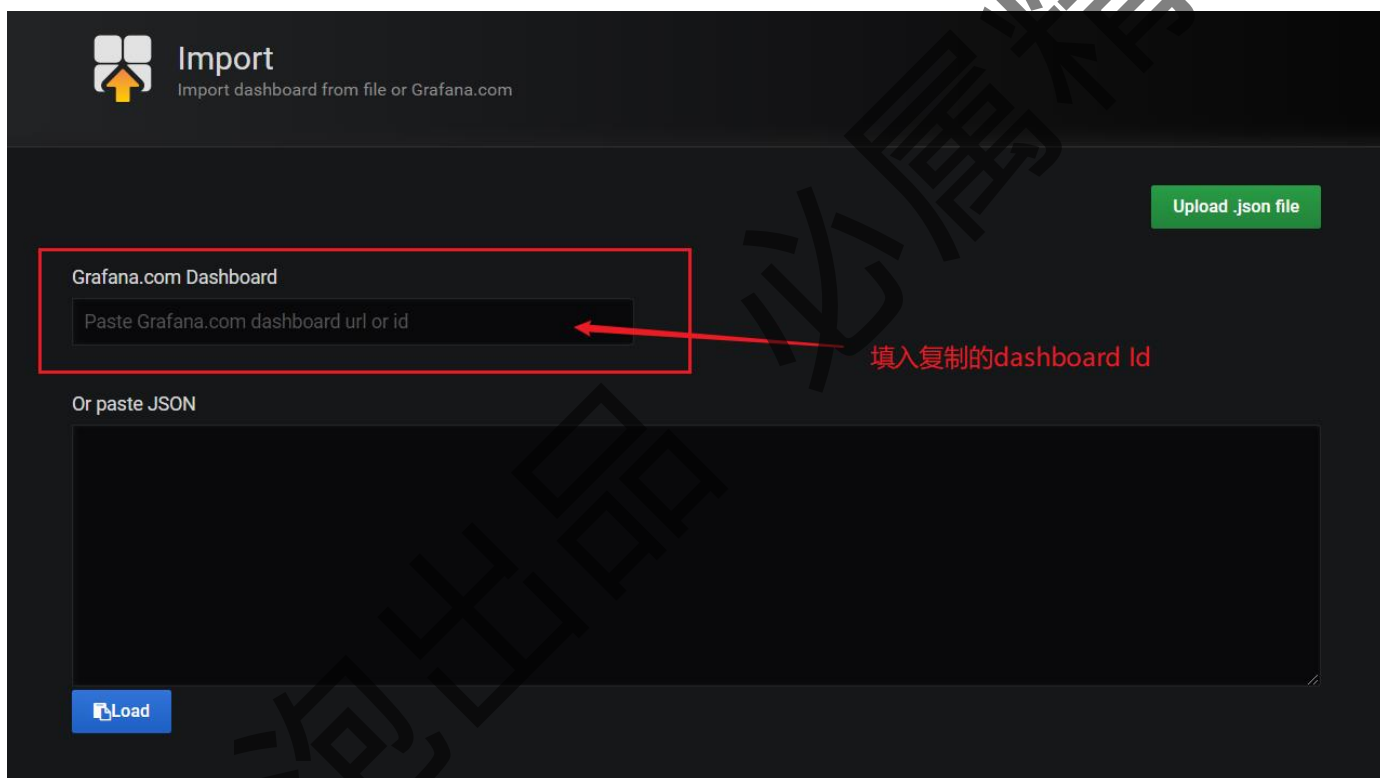
然后打开我们的 Grafana 监控页面，打开 dashboard 的管理页面



点击【import】按钮




然后将我们刚才的复制的 dashboard Id 复制进去



Grafana 会自动识别 dashboard Id 。

然后点击【change】按钮，生成一个随机的 UID，然后点击下方输入框，选择我们之前创建的数据源 Prometheus，最后点击【Import】按钮，即可完成导入。

 **Import**  
Import dashboard from file or Grafana.com

Importing Dashboard from [Grafana.com](#)

Published by	Sébastien Helbert
Updated on	2020-01-17 23:22:32

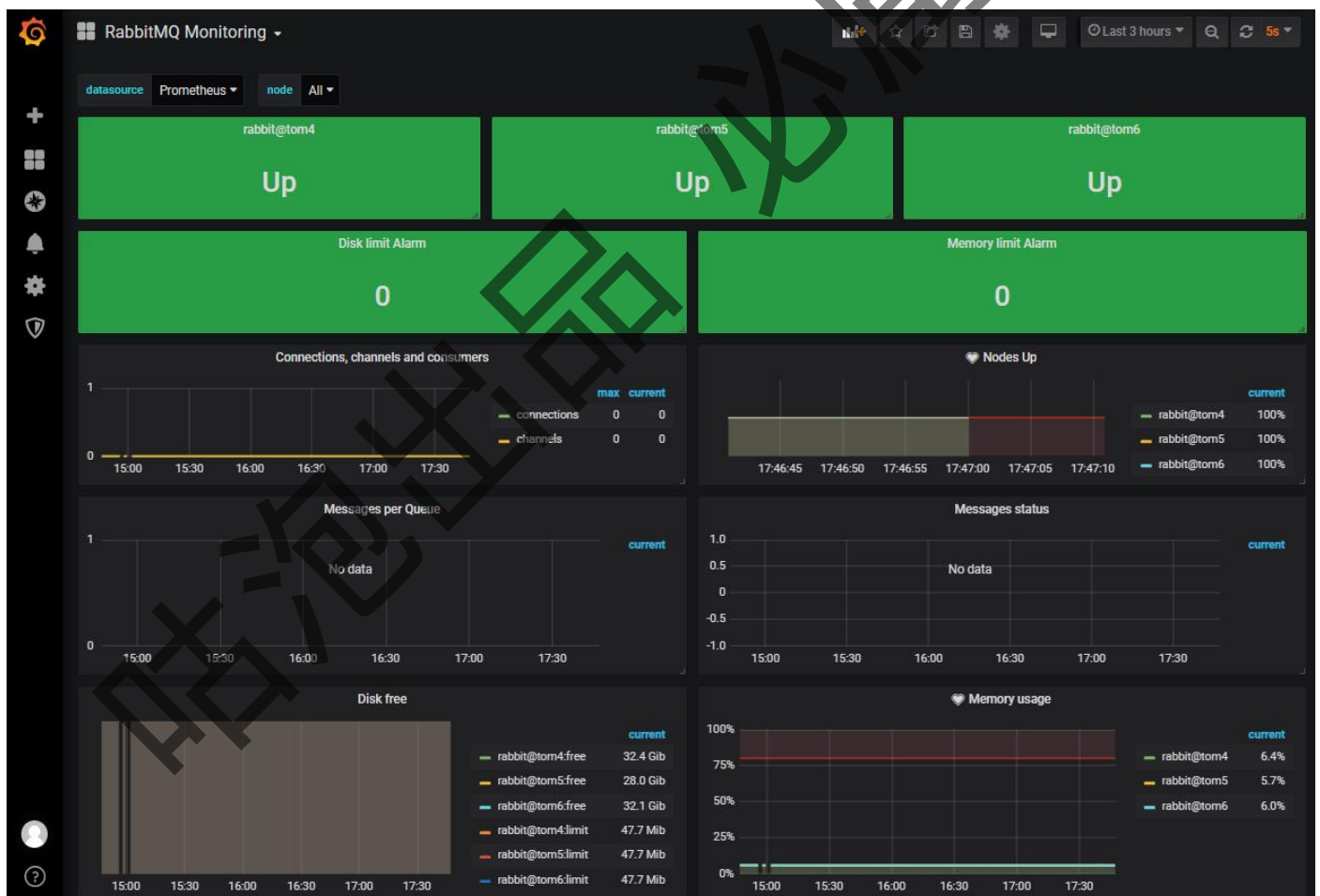
Options

Name	RabbitMQ Monitoring	1
Folder	General	
Unique identifier (uid)	value set	change
prometheus	Select a Prometheus data source	2
	Select a Prometheus data source	
	Prometheus	

3

**Import** Cancel

导入成功后，会自动打开该 Dashboard，即可看到我们刚才设置好的 node 监控



至此 Prometheus+Grafana 安装配置，并监控 RabbitMQ 配置完成。