6张图来自https://springcloud.cc/ 包括了spring cloud现在有的所有组件，以及每个组件的作用。我这里粗浅介绍10个。

spring cloud config

远程配置服务。

远程配置是每个都必不可少的中间件，远程配置的特点一般需要：多节点主备、配置化、动态修改、配置本地化缓存、动态修改的实时推送等。

config允许配置文件放在git上或者svn上，和spring boot的集成非常容易，但是缺点就是修改了git上的配置以后，只能一个一个的请求每个service的接口，让他们去更新配置，没有修改配置的推送消息。而且，如果要根据配置文件的修改，做一些重新初始化操作的话(如线程池的容量变化等)，会需要一些work around的方法，所以建议如果有其他方案，不建议选择spring cloud config。

spring cloud bus

事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化。经常与Spring Cloud Config联合使用。

spring cloud config本身不能向注册过来的服务提供实时更新的推送。比如我们配置放在了git上，那么当修改github上配置内容的时候，最多可以配置webhook到一台config-server上，但是config-server自己不会将配置更新实时推送到各个服务上。

bus的作用就是将大家链接在一条总线上，这条线上的所有server共享状态，当webhook到bus上的某一台server的时候，其他server也会收到相同的hook状态。

但是bus的使用需要依赖于MQ，bus直接继承了RabbitMq & kafka，只需要在spring中直接配置地址即可，但是对于其他类型的MQ，就需要一些手动配置。

最大的问题还是，如果仅仅因为spring cloud bus而让自己的系统引入MQ，显然会有些得不偿失。我理解系统应该在满足现有业务需求的基础上，越简单越好，依赖越少链路越短，越能减少出问题的风险。

eureka

spring cloud的服务发现组件。这个组件讲起来需要大篇幅，最好和consul一起讲。

eureka负责服务注册和服务发现，为了高可用，一般需要多个eureka server相互注册，组成集群。Eureka Server的同步遵循着一个非常简单的原则：只要有一条边将节点连接，就可以进行信息传播与同步。

eureka内部对于注册的service主要通过心跳来监控service是否已经挂掉，默认心跳时间是15s。这就意味着，当一个服务提供方挂掉以后，服务订阅者最长可能30s以后才发现。

service启动连上eureka之后，会同步一份服务列表到本地缓存，服务注册有更新时，eureka会推送到每个service。

eureka也会有一些策略防止由于某个服务所在网络的不稳定导致的所有服务心跳停止的雪崩现象。

eureka自带web页面，在页面上能看到所有的服务注册情况 和 eureka集群状态。

eureka支持服务自己主动下掉自己，请求service的下列地址，可以让服务从eureka上下掉自己，同时service进程也会自己停掉自己。

curl -H 'Accept:application/json' -X POST localhost:${management.port}/shutdown

consul

也是一个服务发现工具，而且自带key-value存储服务、健康检查 和 web页面。

听起来好像比eureka高大上一些，里面使用了gossip协议和Raft协议，但是他的缺点就是比eureka难维护。

服务注册是微服务架构的关键节点。所以我们现阶段选择的是eureka，然后远程配置使用的是spring cloud config。如果要上容器和编排的话，会再看具体情况做选择。

但是，后来发现其实consul提供了官方的docker镜像，直接使用docker-consul集群用户服务发现的话，运维成本会直线下降，后面会考虑把eureka + spring cloud config 换成consul。

ribbon:

客户端负载均衡组件。

服务发现以后，每个service在本地知道自己要调用的服务有多少台机器，机器的ip是什么，端口号是多少，那这个service在本地需要有一个负载均衡策略，为每一次请求选择一台目标机器进行调用，而ribbon做的就是负载均衡策略的选择。

ribbon提供了多种负载均衡策略，包括BestAvailableRule、AvailabilityFilteringRule、WeightedResponseTimeRule、RetryRule、RoundRobinRule、RandomRule、ZoneAvoidanceRule等，没记错的话，默认是ZoneAvoidanceRule。当然，也可以自定义自己的负载均衡策略，比如被调用服务需要灰度发布或者A/B测试的话，就可以在ribbon这一层做自定义。

feign

声明式、模板化的HTTP客户端。

微服务之间的调用本质还是http请求，如果对于每个请求都需要写请求代码，增加请求参数，同时对请求结果做处理，就会存在大量重复工作，而feign非常优雅的帮助我们解决了这个问题，只需要定义一个interface，fegin就知道http请求的时候参数应该如何设置。

同时，feign也集成了ribbon，只要在微服务中依赖了ribbon，feign默认会使用ribbon定义的负载均衡策略。

最重要的是，feign并不是仅仅只能使用在有eureka或者ribbon的微服务系统中，任何系统中，只要涉及到http调用第三方服务，都可以使用feign，帮我们解决http请求的代码重复编写。

hystrix

断路器，类似于物理电路图中的断路器。

正常情况下，当整个服务环境中，某一个服务提供方由于网络原因、数据库原因或者性能原因等，造成响应很慢的话，调用方就有可能短时间内累计大量的请求线程，最终造成调用方down，甚至整个系统崩溃。而加入hystrix之后，如果hystrix发现某个服务的某台机器调用非常缓慢或者多次调用失败，就会短时间内把这条路断掉，所有的请求都不会再发到这台机器上。

如果某个服务所有的机器都挂了，hystrix会迅速失败，马上返回，保证被调用方不会有大量的线程堆积。

Feign默认集成了hystrix。

上面有提到，使用eureka时，当一个服务提供方挂掉以后，服务订阅者最长可能30s以后才知道，那这30s就会出现大量的调用失败。如果在系统里面集成了hystrix，就会马上把挂掉的这台服务提供方断路掉，让请求不再转发到这台机器上，大量减少调用失败。

hystrix执行断路操作以后，并不表示这条路就永远断了，而是会一定时间间隔内缓慢尝试去请求这条路，如果能请求成功，断路就会恢复。

有一点需要注意的是hystrix在做断路时，默认所有的调用请求都会放在一个的线程池中进行，线程池的作用很明显，有隔离性。比如gateway，集成了5个子业务系统，可能其中一个系统的调用量非常大，而另外四个系统的调用很小，如果没有线程池的话，显然第一个系统的大量调用会影响到后面四个系统的调用性能。hystrix的线程池和java标准线程池一样，可以配置一些参数：coreSize、maximumSize、maxQueueSize、queueSizeRejectionThreshold、allowMaximumSizeToDivergeFromCoreSize、keepAliveTimeMinutes等，如果某一个子系统的调用量突然激增，超过了线程池的容量，也会迅速失败，直接返回，起到降级和保护系统本身的作用。当然hystrix也支持非线程池的方式，在本地请求线程中做调用，即semaphore模式，官方不建议，除非系统qps真的很大。

zuul

是一个网关组件。提供动态路由,监控,弹性,安全等边缘服务的框架。

zuul主需要简单配置一下properties文件，不需要写具体的代码就可以实现将请求转发到相应的服务上去。

还可以定制化一些filter做验证、隔离、限流、文件处理等切面，对于网关来说，使用zuul能减少大量的代码。

不过我没有使用过，不太了解，现在我们的网关主要还是基于feignClient、ribbon、hystrix来实现的。zuul默认也集成了这些组件。有兴趣可以研究研究。

turbine

是聚合服务器发送事件流数据的一个工具，用来监控集群下hystrix的metrics情况.

在复杂的分布式系统中，相同服务的节点经常需要部署上百甚至上千个，很多时候，运维人员希望能够把相同服务的节点状态以一个整体集群的形式展现出来，这样可以更好的把握整个系统的状态。

turbine提供把多个hystrix.stream的内容聚合为一个数据源供Dashboard展示.

Spring Cloud Starters

spring boot热插拔、提供默认配置、开箱即用的依赖。

starter 是spring boot框架非常基础的部分。可以自定义starter。

我们现阶段的后台系统中，上述除了spring cloud bus、consul和zuul，其他都使用到。Talking is cheap, github地址: https://github.com/chxfantasy/spring-cloud-demo