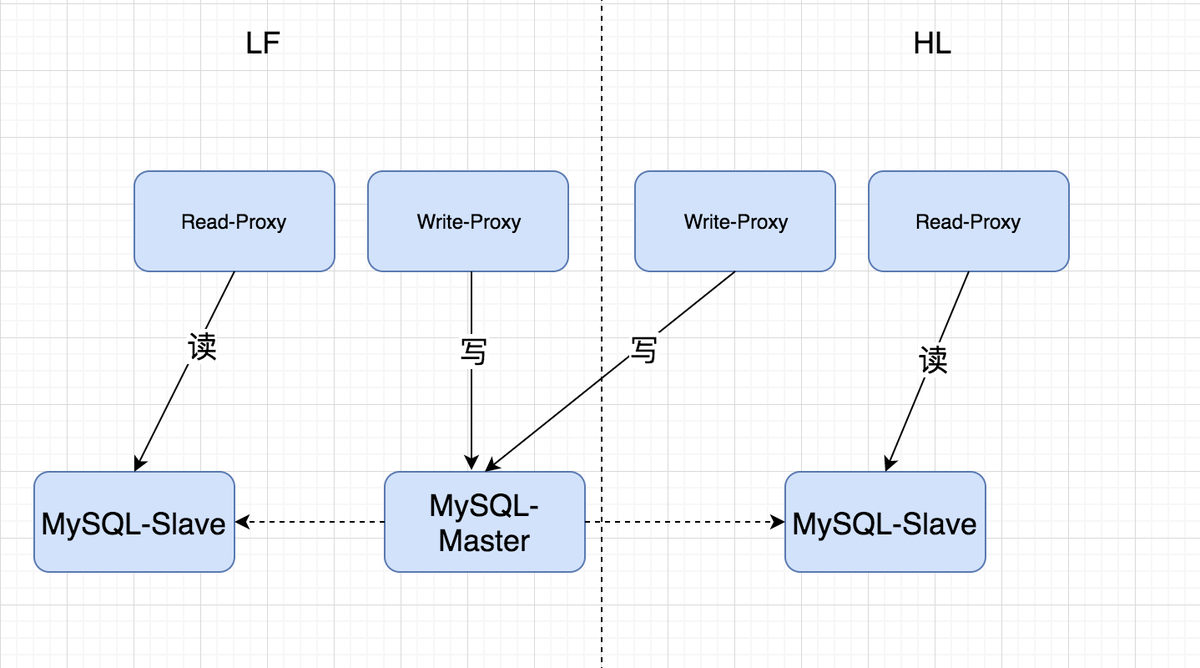
**架构同城双机房方案**

**存储**

**RDBMS**

* 目标
* 日常状态，写入单侧，读取在本机房内完成
* 故障状态下，机房级别容灾。在上层流量调度对应机房时，DB可以将主库切换至流量调度的机房，实现容灾。
* 双机房方案
* 核心架构

1. 

* 核心技术：
* MySQL GTID support:基于GTID，实现autopostion 的获取，降低切换成成本
* MySQL HA solution: 解决双机房之间的切换，最多1min完成单机切换
* 高可用&容灾SLA
* 高可用切换方式：程序判断、报警，DBA手工触发迁移流程，单机群预期在切换时起，最多1min左右完成切换
* 数据durability & consistency：切换瞬间的数据可能有丢失，正常运行时主备数据是eventually consistency
* 依赖分析
* **[进行中]** 高可用组件的多机房部署 - Infrastructure组
* **[进行中]** 业务数据库切换到GTID模式 - Infrastructure + 业务组(被影响方，需要周知，不需要业务做任何操作)
* **[待处理]** Consul在廊坊机房部署，Client通过本机房Consul进行proxy服务发现 - Data-inf组
* **[进行中]** Cannal消费需要更改到GTID模式，保证下游消费数据在数据库切换后正常-Data-platform组
* 时间线
* 6月中旬，Data-platform完成Cannal改造工作
* 6月中旬，完成DB依赖相关服务Down机不可恢复，自恢复的方案及相关脚本或者程序
* 7月中旬，数据库完成核心数据库(要求的范围)双机房的改造工作，HA全量覆盖这部分集群
* 7月下旬，完成部分核心数据库（5-10个）的演练测试工作，保证HA的可用
* **业务双机房灾备需做的调整：**
* 配合架构同学升级数据库为GTID模式。要求业务方：1.提供给DBA升级时间 2.业务方如果能够降级处理的话，最好进行降级处理。
* 业务端需要升级使用架构组提供的Client(Python && Go)，同时确保目前使用的是consul:toutiao.mysql.xxx\_write?ip 这种类似的配置，而不是早期的consul\_dummy的写法，如有老的写法需要和DBA沟通更换成新的方式。
* 业务方需要了解DB的日常模式，写入单侧，读取在本机房内完成；日常级别切换，读写模式不变，机房级别Dwon掉切换后，读写都会在同一个机房内。
* 详细信息
* https://docs.bytedance.net/doc/ZRzkIUhl4dtZYXaZo05ETf
* 机器(包含部分Mongo所需资源)：
* 廊坊所用机器(按类型)：机型：Storage-F，数量：912台
* 怀来已有机器(按类型)：机型: Storage-F，数量：702台
* 怀来额外需的机器 (理想 / 最小)：210台/170台
* 容灾操作方案 - **可以另开文档列举详细步骤，要求非本方向的同学，根据这个方案在获得系统权限的情况下完成操作**
* https://docs.bytedance.net/doc/JopmaNIkpLcAHppDZldnna
* 容灾演练安排 - **要求在8月底之前，规划两次演练**
* https://docs.bytedance.net/doc/JopmaNIkpLcAHppDZldnna

**ETCD**

* 目标
* 在LF和HL任一机房都能读写，机房间保证最终一致
* 双机房方案

1. **方案一，多机房部署强一致 [selected]**
2. 要求至少三机房，其互相之间网络拓扑星状结构
3. 两大主机房(HL, LF) 各部署2实例，第三机房部署1实例
4. 优点：强一致，无需其他改动
5. 缺点：依赖IDC间网络，容易因为机房间的网络抖动导致etcd重新选举影响SLA
6. ~~方案二，双机房双向同步~~
7. ~~通过现有ETCD的watch机制，可以增量同步两机房~~
8. ~~每机房本地均部署从另一机房同步的服务~~
9. ~~优点：跨机房最终一致，对上层透明~~
10. ~~缺点：引入多一个服务，带来运维成本~~
11. ~~方案三，双写~~
12. ~~在现有的组件etcdproxy上，实现双写~~
13. ~~在现有的etcd平台上，实现双写~~
14. ~~优点：容错能力强，实现成本低~~
15. ~~缺点：需要上层服务改动~~

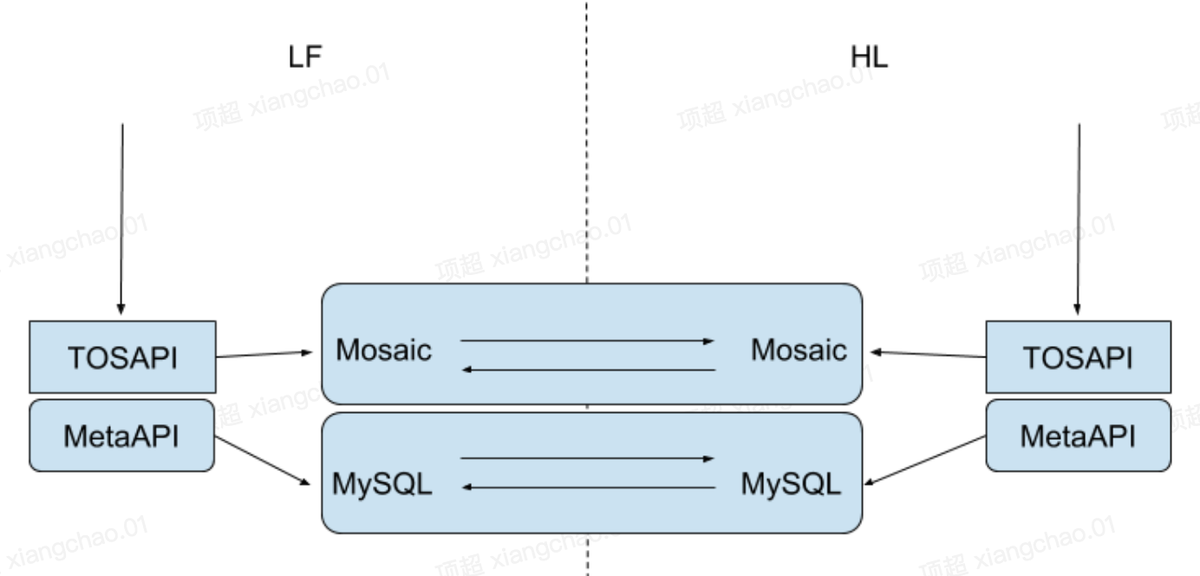
* 依赖分析
* **方案一，要求第三机房直连HL, LF [sys]**
* ~~http://etcd.byted.org 平台依赖 Mongodb [dba]~~ 不对外服务
* **业务双机房灾备需做的调整：**
* 如果已经是通过etcdproxy访问的，无需调整
* 极少部分直接写死 IP 的，要求最终一致的，需要切换到 etcdproxy访问
* 极少部分直接写死 IP 的，要求强一致的，需要更新IP
* 时间线
* 确认方案后，约一周完成

**TOS**

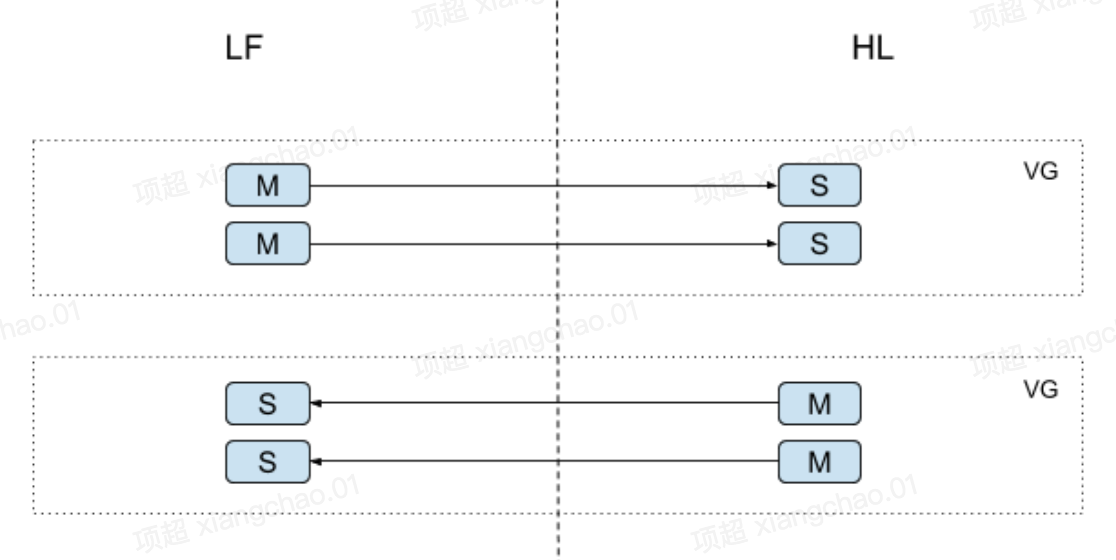
* 目标

1. TOS系统实现在北京lF和HL两个同城机房之间实现双活。
2. 任一个机房断电，另一个机房可以正常服务
3. 两个机房之间专线断开，各自机房仍可以正常服务。
4. 两个机房可以同时提供服务，一致性：保证数据同机房写，同机房读可见。不同机房写和读实现最终一致性

* 双机房方案
* 整体架构



* Mosaic存储引擎双机房

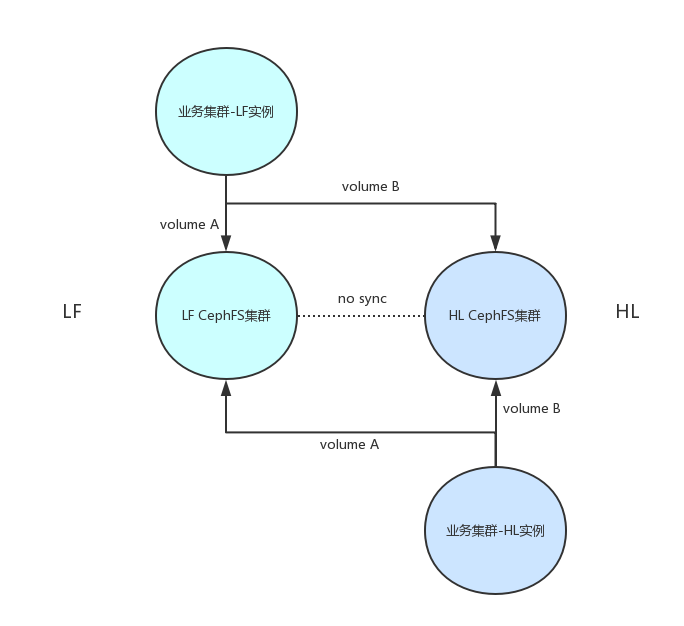


Volume层使用4副本，一个机房两个副本，写入当前机房的两个副本，异步同步到另外一个机房，只要保证两个机房同时具备可写入的VG，就可以保证双机房可同时接收写请求。

* TOS元信息存储双机房
* 元信息存储使用MySQL和TIKV
* MySQL的双机房数据同步采用DRC组件进行数据同步，实现双主数据写入，互相同步
* TIKV的双机房数据同步，由于目前TIKV集群所存储的对象元信息只有一个Bucket，并且此Bucket存储数据可丢失，因此会使用BlobCache作为后端进行存储
* 迁移的SLA保证
* 由于机房之间的数据同步均为异步方式，因此其中一个机房掉电，会造成还未同步的数据丢失。
* 由于是双活模式，因此无需人工干预切换
* 依赖分析
* API层服务部署在TCE，依赖TCE运行实例的双机房容灾功能
* 依赖HL机房API层机器资源到位，预计需要500台均衡B机器加入到TCE中TOS对应的资源池
* TOS依赖的MySQL数据库层的双机房容灾
* 全量升级新版Mosaic的所有集群，LF数据需要同步到HL
* 提供域名下载的相关域名需要SYS支持DNS解析快速切换
* 时间线
* [Done] 6月初，升级所有Mosaic集群，实现数据同步
* [In Progress] 6月中，完成MySQL数据库层的双机房容灾部署
* [Done] 7月初，实现API层的所有代码改造
* 7月中，完成演练和测试
* **业务双机房灾备需做的调整：**
* 业务需要明确了解到在LF和HL部署的服务，会各自访问所在机房的TOS服务。
* 业务上读取和写入TOS需要尽量避免LF和HL两个机房同时操作一个对象的场景，避免非强一致性带来上层影响。
* 访问TOS服务的业务自身需要确保使用了双机房的架构，并且任意一个机房宕机，另一个机房有足够的能力承载相应的流量。
* 业务层避免使用固定的IP进行访问。
* 详细信息
* [TOS双机房方案](https://docs.bytedance.net/doc/NlNH2vJAB7BwX3XWhtpPKg)
* **机器(包含部分Mongo所需资源)：**
* **廊坊所用机器(按类型)：机型：均衡B，数量：558台**
* **怀来已有机器(按类型)：机型: 均衡B，数量：96台**
* **怀来额外需的机器 (理想 / 最小)：462台/300台**

**TBS**

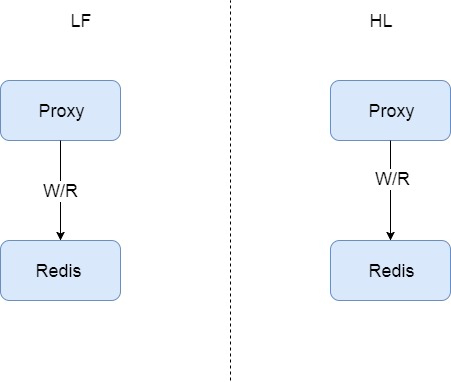
* 目标
* TCE上使用TBS的服务能够做到跨机房容灾能力
* 双机房方案
* TBS提供不同idc的集群供业务创建Volume，TCE支持同时挂载多个Volume，业务集群挂载多Volume，根据需求实现双写，主从等



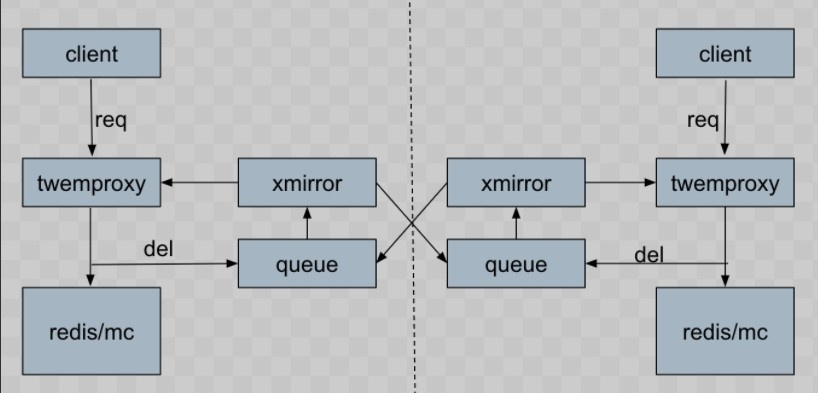
* TBS提供跨idc的集群，满足对性能以及延时不是特别敏感的业务使用
* 依赖分析
* TCE已经支持多Volume挂载
* 推广各个重要业务根据自己需求，实现跨机房容灾
* 时间线
* [In Progress] HL搭建CephFS集群，联系各重要业务，推动业务做好容灾预案，六月底完成
* [In Progress] 搭建跨idc集群，7月底完成
* **业务双机房灾备需做的调整：**
* 双Volume情况
* 申请不同集群的volume，并且升级TCE上面的业务集群，同时挂载双volume
* 针对自己业务的具体容灾情况，修改对volume读写的业务逻辑代码，并增加idc出问题时的降级开关
* 跨idc集群，等待集群ready，将业务迁移到跨idc集群
* 详细信息
* https://docs.bytedance.net/doc/I5tEU57oIeq82cWCEwwZke
* **机器：**
* **廊坊所用机器(按类型)：**Ceph的hdd以及ssd机型均被取消，已和其他业务合并
* **廊坊额外需要的机器(理想)：**storage D/250台，storage-B/20台，cache-B/10台
* **怀来已有机器(按类型)：**Storage-D/150台；Cache-B/8台
* **怀来额外需的机器 (理想)：**storage D/100台，cache-B/10台
* 机器均为理想情况下的需求，对于双机房的事情，只针对tce上的服务做，目前已经挪用资源来填补，**暂时不需要为双机房考虑资源**。
* 容灾演练: https://docs.bytedance.net/doc/gVMIiX0NxUkYrV9TCZ3XXa

**Redis**

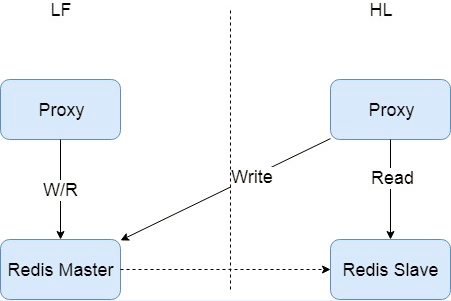
* 目标
* 缓存型和存储型Redis需要实现双机房支持
* 对于缓存型，日常状态，读写都在本机房；故障状态下，机房级别容灾，无需人工干预；
* 对于存储型，日常状态，写入单侧，读取在本机房内完成；故障情况下，机房级别容灾，将主库切换至流量调度的机房，实现容灾；
* 双机房方案
* 场景1 - 缓存性，双机房完全独立
* 机房故障时无需人工干预；
* 在业务未单元化的情况下，机房故障时，另一机房承担所有流量不会导致大量穿透；
* 在业务按照id分流的场景下，所有写请求流量在proxy层同步到对端（架构参考场景2）

1. 

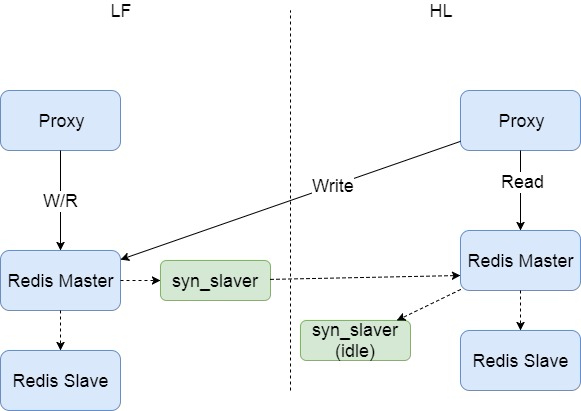
* 场景2 - 缓存性，删除操作弱一致
* 机房故障时无需人工干预；
* 单机房故障下，删除的请求会写入queue，满了之后逐出；（正常状态下删除请求的有效期是30s，因此故障情况下丢弃双删数据影响可接受）

1. 

* 场景3- 缓存型，主从同步
* LF-HL构成一个集群，默认LF为master，HL为slave，基于redis自身的主从机制同步；



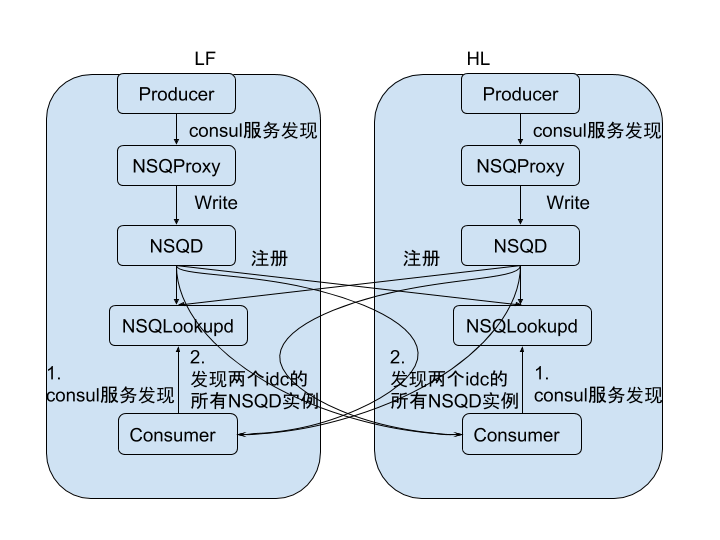
* 支持特大集群，适用老版缓存型redis（一致性hash架构）
* 网络瞬断可能导致全量同步（默认配置100MB的同步缓冲区，超过这个buffer网络还未恢复将全量同步）
* 4 - 存储型，弱一致
* 正常状态下，写到主机房，读本地机房，更新操作通过同步组件同步到从机房；
* 单机房故障状态下，流量切到主机房，同步中断，流量恢复后，数据同步继续（发生单机房切换后，逐步放开集群的同步，避免同一时间全同步导致的带宽打满）

1. 

* 依赖分析
* 平台依赖DB，需部署怀来机房对应数据库服务
* 平台依赖sso登录服务
* 时间线
* [DONE] 6月中，支持机房写流量整体切换，支持任意大规模集群之间的数据同步
* [DONE] 6月底，完成cache 平台 怀来机房部署（包含怀来DB同步服务）
* 7月中，完成遗留服务迁移（遗留服务指：使用不支持机房容灾的缓存架构的服务）
* **业务双机房灾备需做的调整：**
* 使用双机房主从模式部署的缓存架构需切换到独立部署的缓存架构上
* **机器**
* 廊坊所用机器(按类型)：机型：cache 1397、均衡B：495
* 怀来所用机器(按类型)：机型：cache 718、 均衡B：294
* 怀来额外需的机器 (理想/最小)：机型：cache (500台/300台) 均衡B：(300台/200台)
* 容灾操作方案 - **可以另开文档列举详细步骤，要求非本方向的同学，根据这个方案在获得系统权限的情况下完成操作**
* https://wiki.bytedance.net/pages/viewpage.action?pageId=195244487
* 容灾演练安排 - **要求在8月底之前，规划两次演练**
* 详细信息
* https://docs.bytedance.net/doc/16GFKl7DooIqLdVURi119f

**NSQ & TMQ & tPipe**

* 目标
* 支持两个机房分别写本机房，两个机房消费者用同一个channel消费同一个topic的数据
* **注意：不是两边机房的消费者都可以获取同一个topic的所有数据**
* **原因：**
* **业务在两个机房使用同一个channel消费同一个channel**
* **会导致消费者重复消费消息**
* 支持一个idc挂了，流量全部切换到另一个idc
* 支持一个idc的集群挂了，流量全部切换到另一个idc的对等集群
* 双机房方案0



* 依赖分析
* HL机器（512cache机型和均衡B）--sys组
* 业务升级lib（pyutil或者tmq）到最新的版本 --全体使用nsq的业务
* 时间线
* 6月底推动完成web&&ies&&推荐相关集群的业务lib升级
* 7月底推动完成videoarch&&xigua&&applog相关集群的业务lib升级
* 8月中旬推动完成AD相关集群的业务lib升级
* **业务双机房灾备需做的调整：**
* 更新ss\_conf到最新的版本
* 使用最新版本的pyutil/tmq(python/golang)
* 详细信息
* 参考：https://docs.bytedance.net/doc/CqLF0J7nJzixbOMPxwAUOh
* **机器：**
* **廊坊所用机器(按类型)：**
* **均衡B：500**
* **512Cache（加两块1T的SSD）：500**
* **怀来已有机器(按类型)：**
* **均衡B：300**
* **512Cache（加两块1T的SSD）：0**
* **怀来额外需的机器 (理想 / 最小)：**
* **均衡B：200**
* **512Cache（加两块1T的SSD）：500**

**Abase**

* 目标，与 RDBMS 相同
* 单主，常态下所有写，到同一机房，异步复制，读本地
* 发生机房级故障后，切主到健康的机房，分钟级恢复服务
* 现状
* 线上已完全可用
* **业务双机房灾备需做的调整：**
* slave 机房发起的写操作，延迟会略微增加（长传 RTT 2.4ms），因此，需要确保延迟可接受
* 切换时，丢失最新的写入，如不能满足需求请提出（需要开发）
* 业务端需要确保目前使用的是consul:abase\_\*\*\* 这种类似的配置，而不是早期的consul\_dummy的写法。此处更改由Abase同学更改，需要业务方配合重启生效。

**Kafka**

* ~~目标~~
* ~~对于写操作，全部写到主机房（可能跨机房）~~
* ~~对于读操作，分为两阶段完成：~~
* ~~第一阶段，在 Kafka 上面暂不提供透明的双机房方案。如有跨机房消费需求，需要配置 mirror~~
* ~~注意：这期间，只在 LF 提供流式计算服务~~
* ~~第二阶段，从 Kafka 切换到 BMQ，透明支持双机房~~
* ~~现状~~
* ~~写入：已在使用~~
* ~~读取：部分业务已在 mirror~~
* ~~时间线~~
* ~~第一阶段需要持续到至少6月底，这期间会完成 BMQ 的灰度试验，切换少量 topic~~
* **~~业务双机房灾备需做的调整：~~**
* ~~第一阶段，如果需要消费 Kafka：~~
* ~~如果使用流式计算框架 (Flink, Spark)，需要运行在 LF 集群~~
* ~~如果必须跨机房消费，且数据量较大，需要~~[配置 mirror](http://cloud.bytedance.net/inf/kafka/mirror/create?nohash=1)
* ~~如果必须跨机房消费，且数据量较小，可选择跨机房消费~~
* ~~第二阶段，消费 Kafka 的部分对业务透明。但仍要按需考虑资源&部署问题，具体建议请参见YARN部分~~

[kafka双机房新方案](https://docs.bytedance.net/doc/awMONUAW7b7aSag4SeqsSh)

**HDFS**

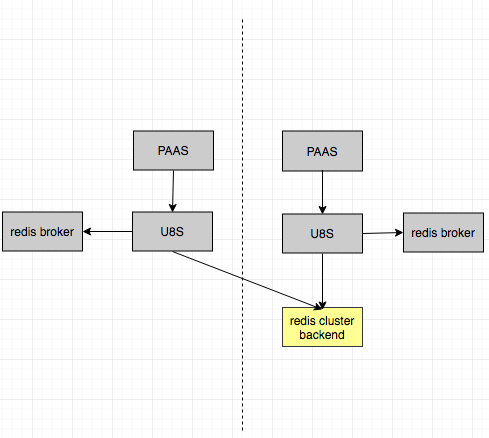
* 目标
* 元数据：单主，继续保证强一致性
* 用户数据：两机房都有副本，如 2:2
* 发生机房级故障后，元数据切主到健康的机房，1小时内恢复服务
* 现状
* 部分业务已在线上使用
* 时间线
* 6月底前，完成数据同步，全量开启
* **业务双机房灾备需做的调整：**
* 需要知晓，在发生机房级别的故障后，HDFS 恢复的预期时间是 1小时。需要根据这个假设来制订容灾方案

**计算**

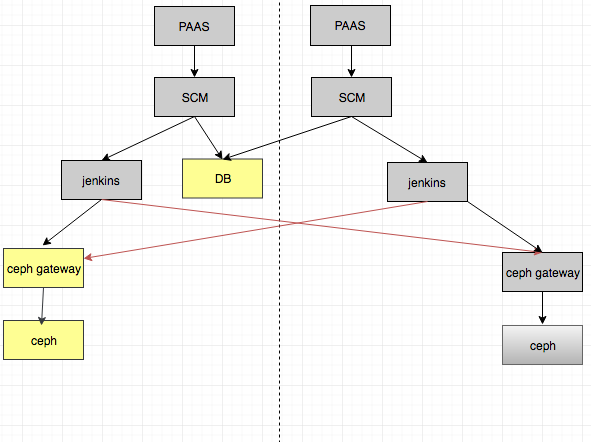
**TCE**

* 目标
* 现在有LF和HL两个机房，希望LF断电一天，TCE服务能正常运行（用户能正常进行上线操作）
* 目前希望LF和HL两个机房达到active-standby这种状态，希望在分钟级内完成切换
* 详情：
* https://docs.bytedance.net/doc/Cxx4WENYjT41DoZHsV9ief
* 双机房方案

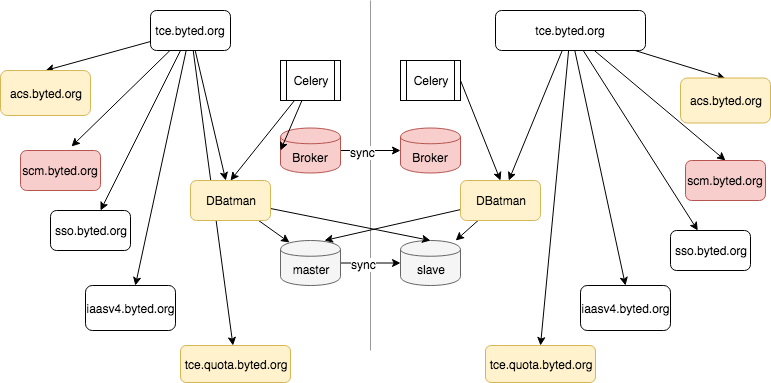
u8s:

1. 

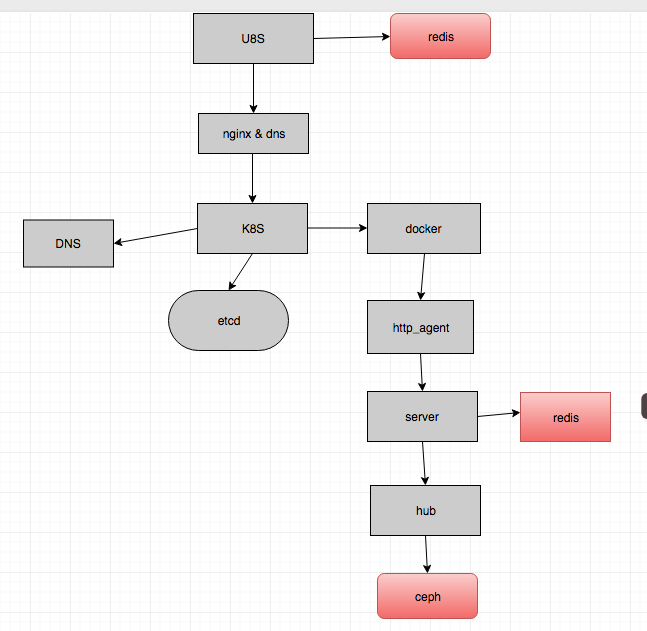
scm:

1. 

paas

1. 

* 依赖分析

1. 

* 时间线（还有5周）
* 第一周（本周）
* 确定双机房最终方案
* 第二周
* 数据库迁移到proxy
* pass
* scm
* acs
* quota
* tce quota改造成依赖u8s @杨润
* redis mq迁移到rabbitmq
* u8s完成redis改造
* http\_agent完成redis改造
* 第三周
* 完成降级方案等容灾改造
* scm改造完成
* 第四周
* 完成环境部署，测试
* **业务双机房灾备需做的调整：**
* 把实例迁移到另一个机房
* 确保冗余机房有足够的quota
* 详细信息
* **机器(理想，最小)：16000 / 2000 台均衡**
* **容灾操作方案:**
* u8s:
* 登陆到开发机，git clone git@code.byted.org:tce/tce\_u8s.git;
* $tce\_u8s/script/switch2hl.sh
* k8s相关组件
* no action

**MS平台**

* 目标
* MS平台在LF和HL两个机房分别部署达到active-standby这种状态，希望在分钟级内完成切换
* 双机房方案
* 域名双机房都可访问
* 前端双机房部署，日常可同时访问
* 后端双机房部署，日常可同时访问
* 数据抓取服务双机房部署
* bosun报警服务双机房部署，正常双机房各跑一半的规则，可分钟切换全量规则到某一机房
* 依赖分析
* mysql/ redis/ etcd/ OpenTSDB / Grafana / ee的发送lark报警接口/ sys打电话服务 (低优先级)
* 时间线
* mysql/ redis/ etcd 双机房部署
* ms平台前后端与数据抓取服务双机房部署，一处机房异常，不会对平台服务造成影响
* bosun改造为双机房各部署一半的规则量，当某一机房出现异常，能够利用开关自动切换为某一地区全量
* **业务双机房灾备需做的调整：**
* 无
* 详细信息

**服务治理系统**

* 目标
* 单机房挂掉的时候，另外一个机房配置仍能正常读写，服务治理功能正常
* 暂时不考虑自动切换流量策略，那样会涉及到更系统的报警、容量预估、切换策略等
* 双机房方案
* 两机房各自部署etcd集群，ms平台两边部署。因为服务治理的配置没有太强一致的需求，所以进行配置修改的时候双写etcd，平台写入的时候少量重试。
* 写入失败的时候把失败信息用MQ记录，利用etcd的cas操作补数据，防止误覆盖平台的后续正常修改
* 依赖分析
* 依赖etcd的双机房部署
* 依赖ms平台无状态服务的双机房部署
* 依赖mq和补数据服务
* **业务双机房灾备需做的调整：**
* 时间线
* 6月15日双写ready
* 6月22日补数据ready

**资源调度 YARN**

* 目标
* 在每个机房提供独立集群，按需部署
* 双机房情况下，总算力没有冗余；三机房情况下，总计算资源两倍冗余
* 任意机房故障：双机房情况下，做到降级运行；三机房情况下，做到无损运行
* 现状
* HL 集群已 ready
* 详细方案：https://docs.google.com/document/d/1t-MTIzQivOFaLbexZCtdVbdH13WGFf7NzvJsuoYuz6w/edit
* **业务双机房灾备需做的调整：**
* 明确哪些计算任务是不可降级的，并有预案做到优先恢复

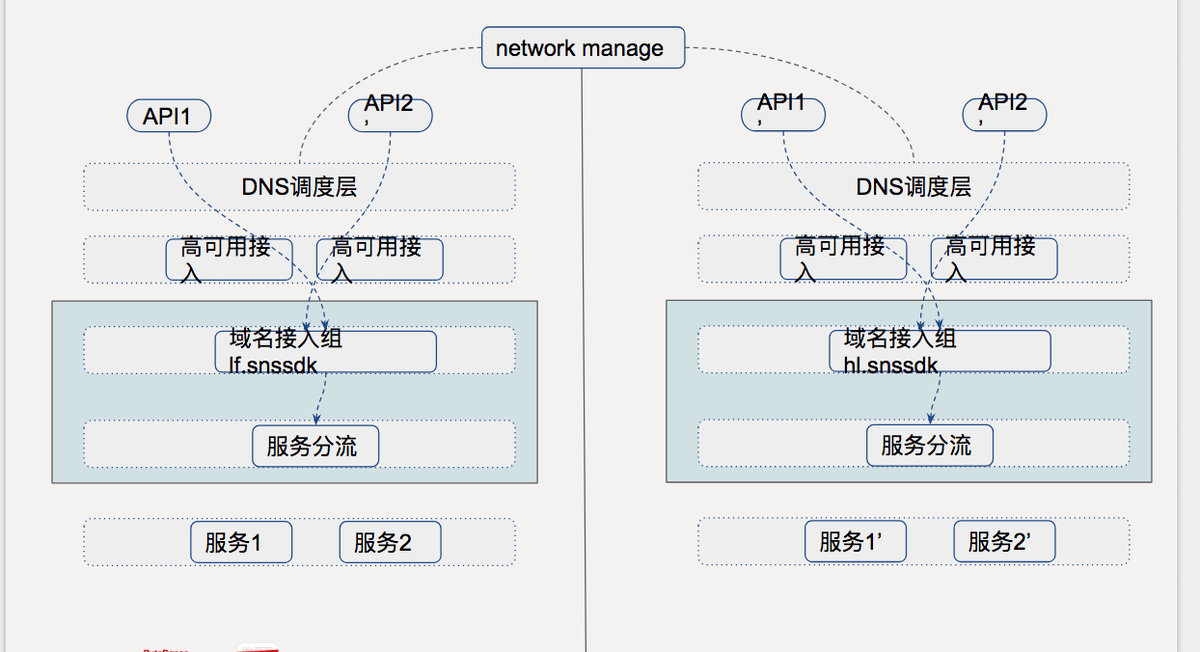
**流式计算平台 Dayu-TSS**

* 目标
* 平台本身做到容灾，任务需要用户手动拉起
* 现状
* 未实施
* 方案：https://wiki.bytedance.net/pages/viewpage.action?pageId=175559519

**网络**

**TLB**

* 目标
* 双机房方案

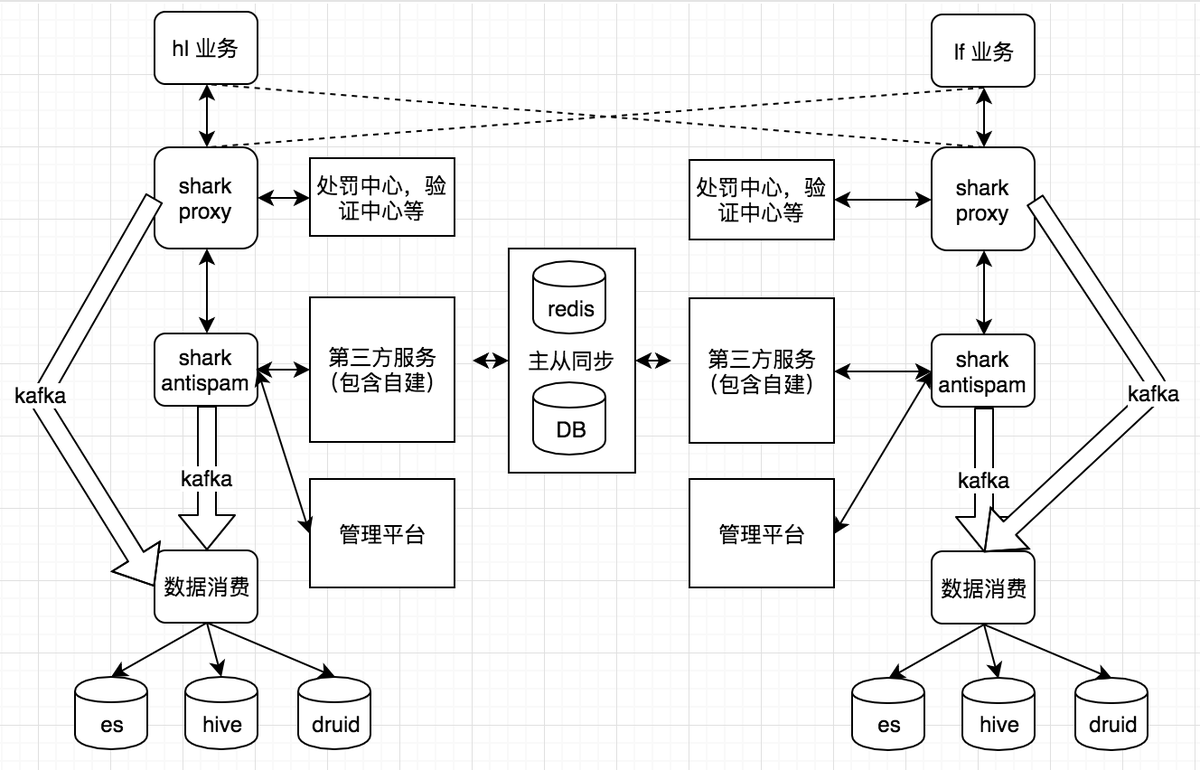
1. 

* 双机房流量调度方案:
* 通过域名进行机房级别的调度:
* 所有域名 在HL都有与之对应属性的新域名;
* 通过httpdns控制 用户的 访问域名;
* cdn根据用户的访问域名，回源至相应的机房;
* 示例:
* 现有域名lf.snssdk.com is.snssdk.com是lf流量的入口;
* 新申请域名:
* hl.snssdk.com 与 lf.snssdk.com对应, 是hl直接回源流量的入口;
* il.snssdk.com 与 is.snssdk.com 对应，cdn动态加速域名的流量入口;
* 域名间流量调度的机制:
* toutiao.network\_manage.api服务根据did 对指定特征的用户返回hl;
* 双机房配置管理方案:
* 方案1:
* 总体描述: 双机房的配置无差异，一个region的多机房的域名配置完全相同
* 实现方案:
* tlb平台实现同一份配置在两个机房设备同时上线，保证一直强一直
* 业务双机房
* 方案2:
* 双机房的配置差异化，每个idc独立管理自己的域名列表;
* 每个idc机房的
* 依赖分析
* 双机房机房入口依赖:
* httpdns:
* 端上域名配置服务: toutiao.network\_manage.api
* 域名下发服务情况:
* 在tce双机房部署，
* 该服务不依赖于任何其它组件和服务，数据输入从配置文件;
* 机房故障需要彻底切换域名时，依赖于该服务配置在tce上线配置;
* 端上容错机制:
* 机房域名故障时，依赖于端上自动容错至备用域名;
* cdn动态加速容错机制:
* 机房故障时，cdn能够快速切换至backup机房回源; 需要sys确认;
* 内部服务dns依赖:
* 公司dns现在已经有双机房容灾部署，可以dns解析正常;
* 目前dns切换没有热切机制，冷切dns缓存时间默认在10min，时长较长;
* TLB 平台依赖:
* tlb\_platform:
* 平台无状态，默认采用双机房部署;
* mysql
* db需要主从，默认写主机房，读取从机房
* redis
* 目前config meta数据是已经单元化的，每个机房的LB config meta都是存储在机房内的redis;
* celery
* 双机房独立部署已经支持
* tlb\_agent
* 无状态，local部署，无跨机房风险;
* 机房故障时，提供快速导入服务IP列表的接口;
* tlb\_watch
* 无状态，local部署，无跨机房
* ;
* naming:
* 依赖于naming的双机房方案;
* consul:
* 依赖于consul的双机房方案;
* 时间线
* 6月底申请设备，开始新建双机房的TLB集群
* **业务双机房灾备需做的调整：**
* sys对于双机房域名策略方案，暂时没定
* 域名策略方案，业务需要有感知新域名跟原来老域名的映射关系;
* 端上完全是没法进行服务级别的分流;
* HL现在启动了默认流量过程;
* 默认HL流量需要修改为本机;
* 业务单独开启HL的流量
* 服务dns解析没有双机房属性的 @郑玉虎
* **资源需求:**
* **lf所用机器均衡B: 340台**
* **hl已有机器: 0**
* **hl需要机器（理想/最小）: 340台 / 200台**
* 详细方案:

**业务中间件**

**反作弊**

* 目标
* 同城双机房部署对等服务，同时留出足够的buff，保证其中一个机房挂掉的情况下，另一个机房可以承载整体流量。
* 双机房方案



* 在线服务双机房部署，正常情况下，业务调用shark是本机房调本机房，当出现某个机房服务挂掉的情况下，框架内部自动fallback到另一个机房或者通过服务化平台手动切到另一个机房。
* 管理平台及自建第三方服务，底层数据依赖db和redis的主从同步，当其中一个机房的存储挂掉之后，可以通过proxy走到另一个机房，如果挂掉的是主，那么需要proxy那边发生主切换，而服务调用则和上面一致。
* kafka的写依赖data那边的broker自动做双机房容灾（hive和druid的写本质上是kafka的写），而数据消费的kafka读则在双机房部署相同group的consumer
* es目前是依赖的域名去进行读写，依赖tlb的双机房容灾，底层数据采用双副本双机房部署方式
* 依赖分析
* 在线服务依赖流量切换
* 底层存储依赖db和redis，es的主从同步
* 消息队列依赖kafka本身的读容灾
* 时间线
* 6月上旬开始部署双机房在线服务和消费脚本，并确认流量切换的方式
* 6月中旬开始部署底层存储相关的组件
* **业务双机房灾备需做的调整：**
* 业务方只需要升级到最新的框架，并且机房的调用不要在代码里写死，能通过ms进行切换就行，在正常情况下，本机房调本机房，当其中一个机房挂掉之后，可以通过ms切到另一个机房
* 详细信息

**反爬取**

* 目标
* 同城部署LF,HL两套机房，互为备份。当某个机房挂掉时，流量可以自动切到另一个机房，每个机房的容量足够全部流量。
* 双机房方案
* 反爬取依赖离线YARN上的Flink任务，kafka消息队列，DB，Druid，ES，Redis等。
* DB，Redis等存储关键的数据和状态，同时需要一致性，在双机房均部署，master在LF，slave两个机房各一份。当LF机房不可用时，设置HL的slave为主。
* Flink任务在LF、HL均部署一份同一个consumer group消费同一个topic。
* ES本身支持透明的双机房部署，不同的副本分配到不同的机房。Druid在资源足够情况下也支持透明地切机房（带来一些慢查询和不可用）。
* 接入服务部署在两个机房，具体上游流量的切换（自动/手动）需要服务治理框架支持。
* 依赖分析

1. Flink的YARN在HL的部署；
2. DB,Redis,Kafka的双机房方案落地。
3. ES的双机房方案已经ready，druid ready。
4. 服务框架的服务发现的双机房容灾方案落地。

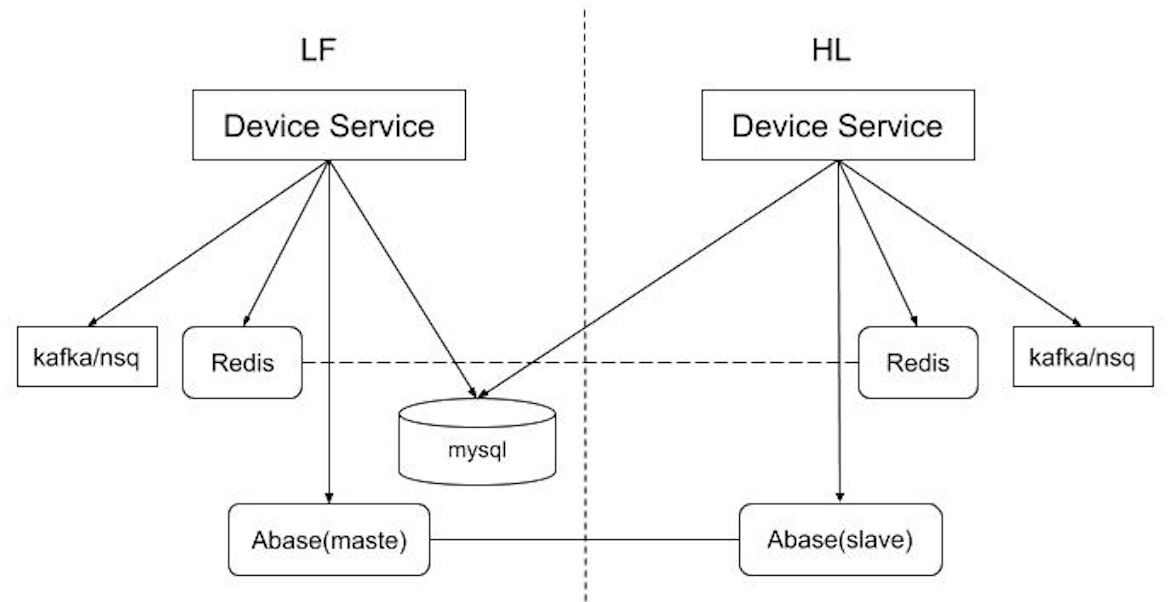
* 时间线

1. 6月上旬在HL部署接入服务与数据库从库。
2. 其余依赖上述方案的落地。

* **业务双机房灾备需做的调整：**
* 使用Middleware的业务不需要做代码调整，最好升级框架和middleware版本到最新；rpc调用的同学需要注意不要指定机房。
* 事故发生时，需要在ms操作切换机房到备用机房。
* 自动切换需要依赖框架。
* 详细信息

**Device Info**

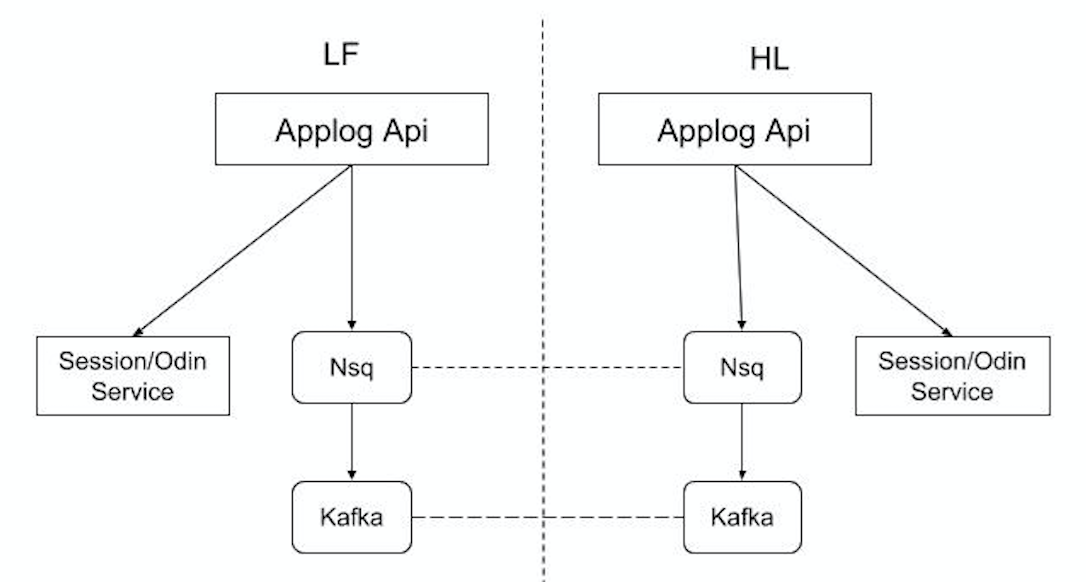
* 目标
* 两个机房都可以提供服务，一个机房挂掉后另一个机房可以承载所有流量
* 双机房方案

1. 
2. 由于Mysql准备迁移新库，所以目前没有在HL部署从库，但是DB迁移近期内完不成，如果要支持双机房还需要DBA在HL部署从库，提供DB层面的双机房支持

* 依赖分析
* 所依赖组件DB/redis/abase/kafka/nsq 的双机房支持
* 时间线
* 6.15 DB在HL机房部署从库
* 待其余依赖组件的落地
* **业务双机房灾备需做的调整：**
* 详细信息
* **机器：**
* **依赖TCE资源**

**App Log收集**

* 目标
* 两个机房都可以提供服务，一个机房挂掉后另一个机房可以承载所有流量
* 双机房方案



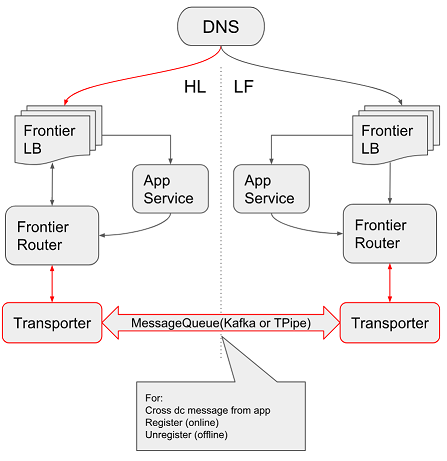
* 依赖分析
* session、odin rpc服务的部署(已部署，但容灾方案是否可行待确定)
* 依赖组件 nsq、kafka 的双机房支持
* 时间线
* 待依赖组件部署完成后，applog api即可部署
* **业务双机房灾备需做的调整：**
* 详细信息
* **机器：**
* **廊坊所用机器(按类型)：机型：均衡B，数量：163台**
* **怀来已有机器(按类型)：机型：均衡B，数量：0台**
* **怀来额外需的机器 (理想 / 最小)：163台均衡B/ 台**

**端监控**

* 目标
* 两个机房都可以提供服务，一个机房挂掉之后不影响整体服务
* 双机房方案
* API层：日志收集服务，配置下发服务双机房同时部署，需要保证单机房的资源能够撑住全流量
* ETL任务：目前部署在Flink，需要inf来提供容灾方案
* 实时分析
* Druid：所有节点双机房部署，历史节点在LF，HL分别保存一个副本(目前机器资源基本都在HL，需要申请资源)
* ClickHouse
* 方案一：利用官方提供的基于zk的备份方案，LF和HL同时部署一个集群，一主一从的模式，不过这种模式会浪费一半的机器资源，同时还不确定开了replica之后对实时数据的写入吞吐有没有影响，需要验证
* 方案二：文件存储挂载tbs，数据的容灾通过tbs保证，clickhouse-server双机房部署，挂机房的时候需要人工介入，在另外一个机房启动对应的server
* 依赖分析
* API层服务依赖MySQL，Redis，NSQ
* 流式计算任务依赖Yarn
* Druid依赖zk，hdfs，MySQL
* ClickHouse可能会依赖zk和tbs
* 以上组件都需要提供双机房支持
* 时间线
* TBD
* **业务双机房灾备需做的调整：**
* 详细信息

**Frontier长连接**

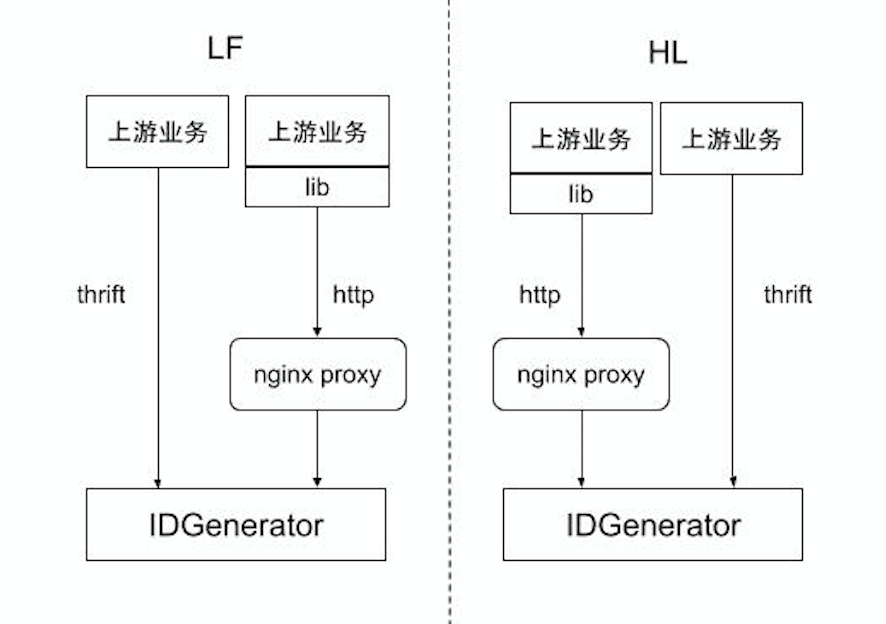
* 目标
* 双机房可以同时接收连接
* 双机房可以互相发送实时数据
* 双机房方案
* 如下图，红色部分为TODO。
* toutiao.frontier.frontier 服务简称 Frontier LB
* toutiao.frontier.backbone 服务简称 Frontier Router
* 接入的业务逻辑部分，称之 AppService

1. 

* 对于本机房的连接，Router即可直接转发到具体连接上，即上图的本机房Frontier LB层；
* 对于跨机房的连接，Router会写到下层MQ，最终转发至目标机房Router；
* 上图没有存储路由信息的redis，因为对redis的跨机房方案没有依赖，可以省略；
* 服务功能的tradeoff:
* 本机房的连接，因为是同步调用Frontier LB，能返回给AppService连接是否真实存在；
* 对于跨机房的连接，因为是异步调用，不保证最终调用是否成功。也可能因为路由信息已经过期，实际连接早已下线。前面两者无法返回给AppService；
* 依赖分析
* ~~HL外网接入~~ **ttgw接入**
* MQ跨机房双向方案
* **业务双机房灾备需做的调整：**
* 每机房部署 AppService (即Frontier中的Backservice)
* 确保 AppService 依赖的存储均已准备好双机房解决方案
* 时间线
* 6月中旬完成整体开发
* 6月下旬可上线(HL外网、MQ等已经解决的前提下)
* **需要资源**
* TTGW: 8\*均衡D @杨兴武
* FrontierLB: 16\*Cache(40C 512G 1T). (需要打上ttgw的toa patch依赖)
* 无状态服务: TCE实例 - 25 \* 4 cores
* 消息队列: 能支撑 15万QPS

**IDGenerator**

* 目标
* 上游服务调用本机房服务，两个机房都可以提供服务，一个机房挂掉后另一个机房可以承载所有流量
* 双机房方案



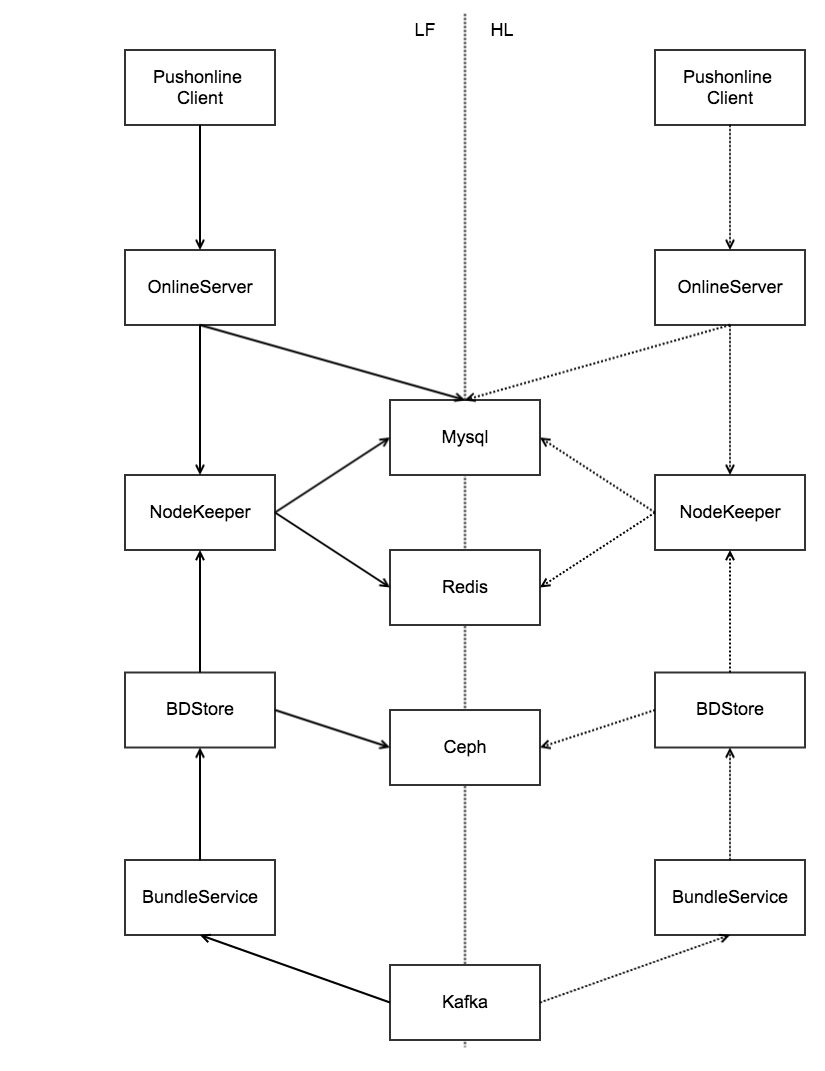
* 依赖分析
* 依赖机器资源(6台均衡B)
* 时间线
* 6.18 idgenerator在hl机房部署
* 6.20 idgenerator 库升级
* **业务双机房灾备需做的调整：**
* 业务需要升级库到最新版本(go/py)
* 详细信息
* **机器：**
* **廊坊所用机器(按类型)：机型：均衡B，数量：8台**
* **怀来已有机器(按类型)：机型：均衡B，数量：0台**
* **怀来额外需的机器 (理想 / 最小)：8台 / 6台均衡B**

**Data持续集成双机房方案**

这个双月，Data的持续集成架构会与SCM做整合（详细方案参见https://docs.bytedance.net/doc/1if3aGwTQFD1emxqtMuAjg），双机房方案也会一并考虑，这里只说明独立于SCM的上线部分，也就是Pushonline相关部分。

**Pushonline**

* 目标
* 同城部署LF,HL两套机房互备。当某个机房故障时，流量可以全部自动切到另一个机房，每个机房的容量足够除故障机房外全部流量，实现分钟级别恢复服务。
* 当两个机房间无法互通时，两机房内部各自上线运行正常，总体上线不受影响。
* 当前正在运行的上线任务服务恢复后执行进度不受影响。
* 双机房方案
* 如下图，实线为当前架构，虚线为双机房架构
* online-server、node-api-server双机房部署，域名、consul双机房可访问
* node-api-server设置开关完成故障时redis代理切换
* bundle service、bdstore增设hl冷备，cdn设置通过脚本切换bdstore，故障时启用冷备bdstore节点
* bundle service设置开关，故障时切换kafka代理
* bdstore发布产物数据从主节点异步同步至冷备节点
* 故障时node-api-server、bundle\_server和cdn需重启，预计1分钟恢复访问



* 依赖分析
* redis：存储型，弱一致，增设hl实例，故障时切换至另一个机房
* MySQL：数据库层的双机房容灾
* tlb\_platform：默认采用双机房部署
* Git：依赖git支持双机房
* **业务双机房灾备需做的调整：**
* ByteOS增加bundle service冷备访问机制@郭士伟
* 时间线
* 确认方案后，约一周完成
* TODOs
* OnlineServer/Nodekeeper的Mysql目前是两个库，计划合并之后配置双机房容灾。@刘哲
* Redis配置多机房容灾。@刘哲
* HL服务搭建。@刘哲
* HL部署开发环境。@周鸿斌
* SCM改造完成。@周鸿斌

**编译集群及预上线**

**1.jenkins集群**

1.切换jenkins 工作目录到ceph。

2.在hl部署jenkins集群，两个集群互为备份，某个机房不可用之后手动启动另一个机房。

**2.预上线集群**

由于预上线强依赖于引流，机房故障后引流源机器down导致没有流量来源，如果引流都设置两个机房双份流量工作量比较大，而且预上线并不是研发流程中都强依赖，预上线系统需要增加降级开关，故障开始后停止预上线。

服务器供应情况:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 均衡B | 300+200+2000+6+100+180 | TOS+redis+TCE+IDGenerator+APPLOG+yinping | HL 库存 均衡B 500左右 | 先按照均衡B算数量 最后数量可以需要均衡B 均衡D 均衡E 混合 |
| DB-A | 100台 | 数据库张博 | 满足概率>90% | 数量与张博单独确认过 |
| Cache-B(A) | 10+300 | TBS+redis |  |  |
| Storage-A | 100+20 | NSQ+yinping |  |  |
| 均衡E | 33 | yinping |  |  |
| Storage-B | 450 | abase | HL 库存394+80+100 |  |

TLB: @郑玉虎 与@黄龙 单独沟通, 系统部支持, 这个是特殊需求不在标准供应中探讨

需要调拨HL 并上架:

DB-A: 100 节点(10)

Storage-B: 100 节点(3)

Cache-B: 320节点(10)

Storage-A: 120节点(4)

均衡E: 800 节点(其余)

当前成功率75% 以上

设备数量在6.28日封版未来不做变更.

**国内双机房**

**缓存双机房-单元化场景**

【目标】

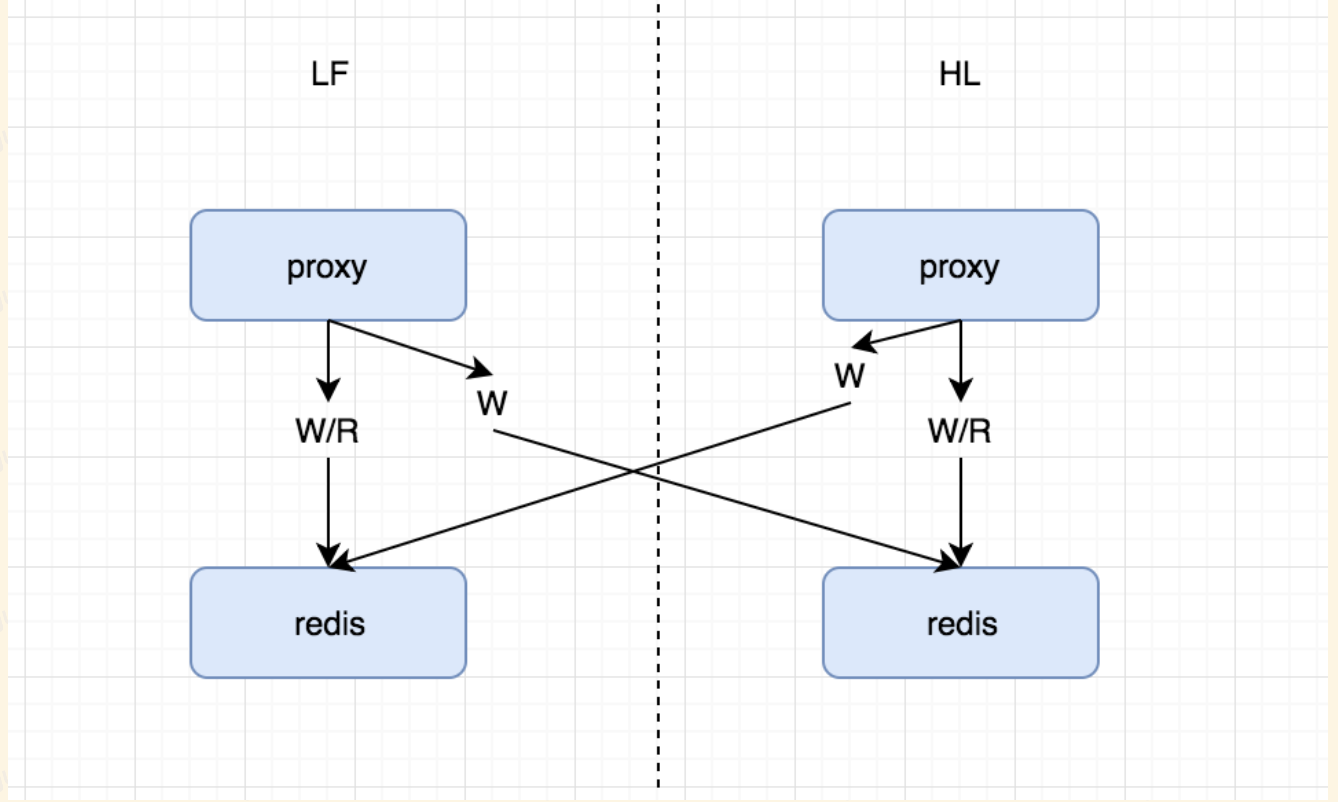
针对单元化后的缓存场景，当一个机房挂掉后，另一个机房能承担全部流量，且无请求穿透（不会因为单元化而造成的cache穿透）。

最终一致性，通过key过期回源达到最终一致性。

与双删区别：在单元化场景下，两个机房的请求相对独立，cache只能cache住一部分数据，当一边机房挂掉时有数据穿透风险，双机房单元化场景下暂时考虑双写方案规避穿透风险。

【方案】

proxy层做两边机房双写，如下：



1、同步写请求到自己机房，异步写请求到对端机房（机房间2ms延时），返回结果为写自己机房的返回

2、双写之前是否需要先保证两边cache数据基本一致？按需要选择

【一致性】

假设在双写前两个机房数据是一致的

直接更新（set、incr、sadd、hset、zadd）→  双写，如果双写前是一致的，则双写后两边数据也一致

先del再回源  →  双删+双写，由于写对端是异步的，到对端可能有顺序性问题，但是因为回源需要一定时间，应该可接受；

造成不一致的场景：

逐出 → redis自发逐出，不会同步到对端机房，可能造成一边有这个key，一边没有。

过期 →  一边过期了，另一边还未过期，造成一边有这个key，一边没有。

update操作（如incr、zadd等）异步写失败，可能造成两边都有，但value不一致

结论：本机房内能保证cache的数据的一致性，但不能保证本机房的数据在对端机房的数据一致；因此，当机房故障靠另一边扛流量时，可能有数据不一致的情况。

这个方案的使用前提是业务进行了单元化，是给单元化后的缓存类型集群灾备使用（对上述set等一致性要求不高），对于一致性要求高的建议使用单主架构；