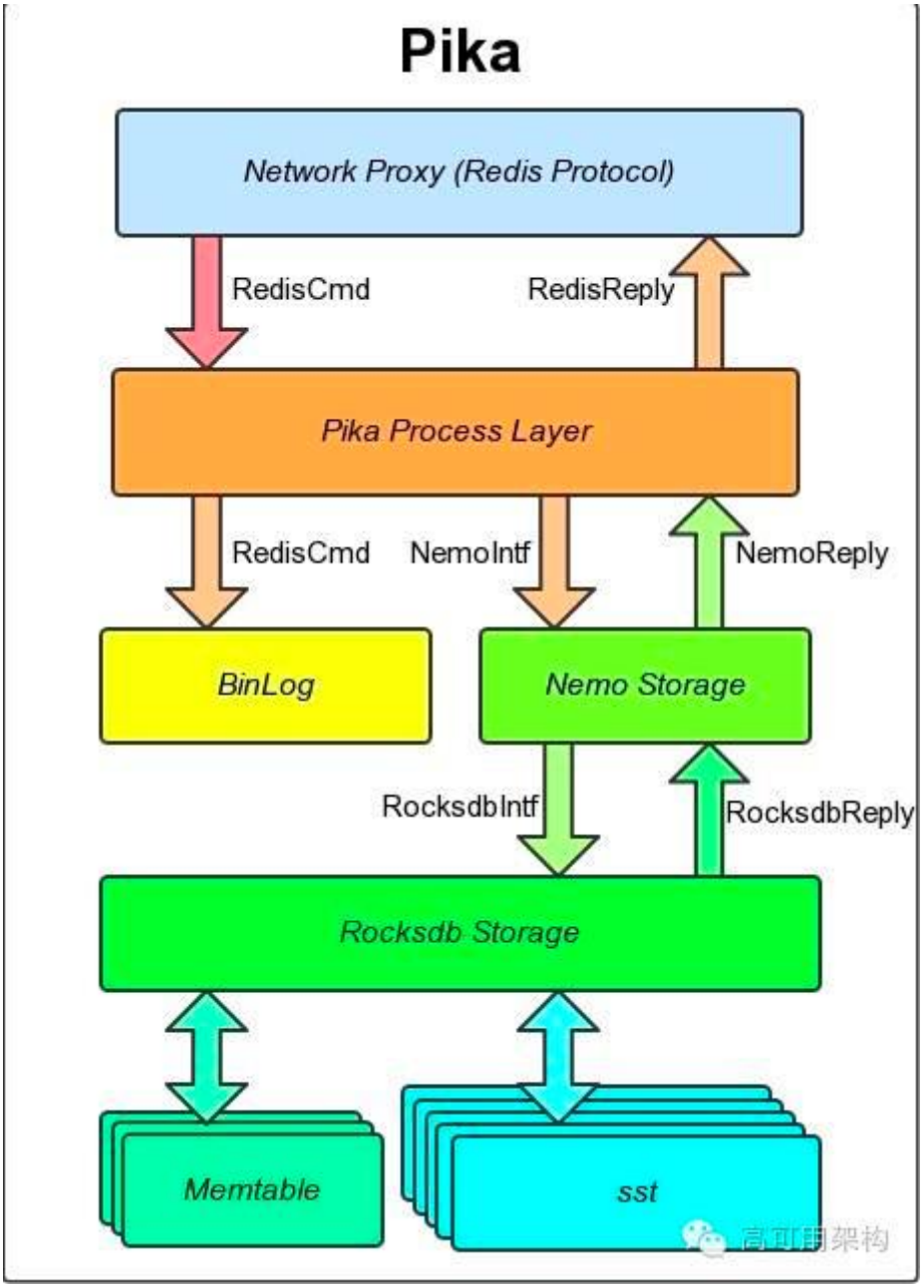
pika serverless 设计方案

现状

架构



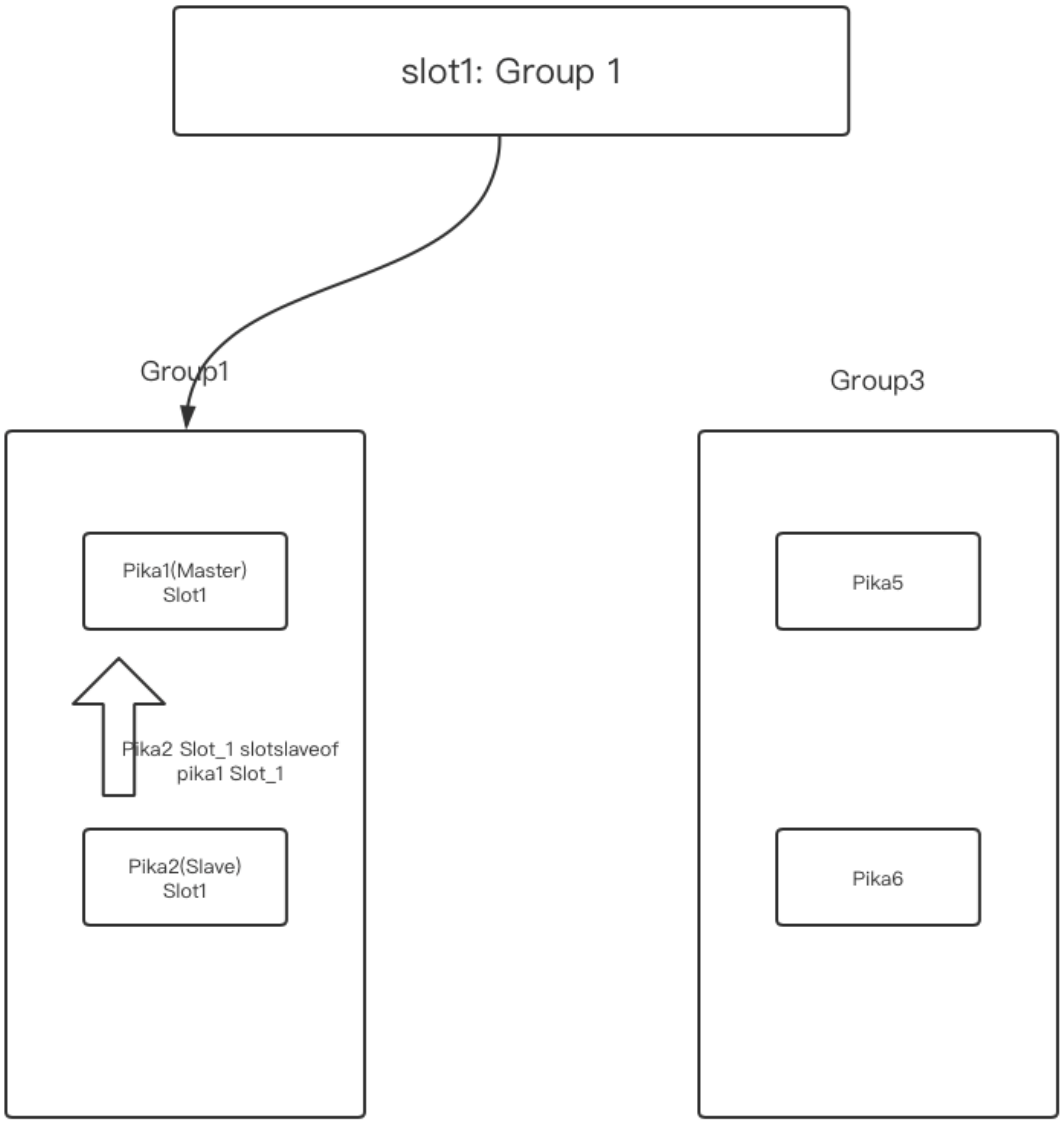
集群架构

目前 pika 还不支持分布式数据库，可以使用主从模式，或者使用中间件代理搭建集群，做数据分片

主备

分片

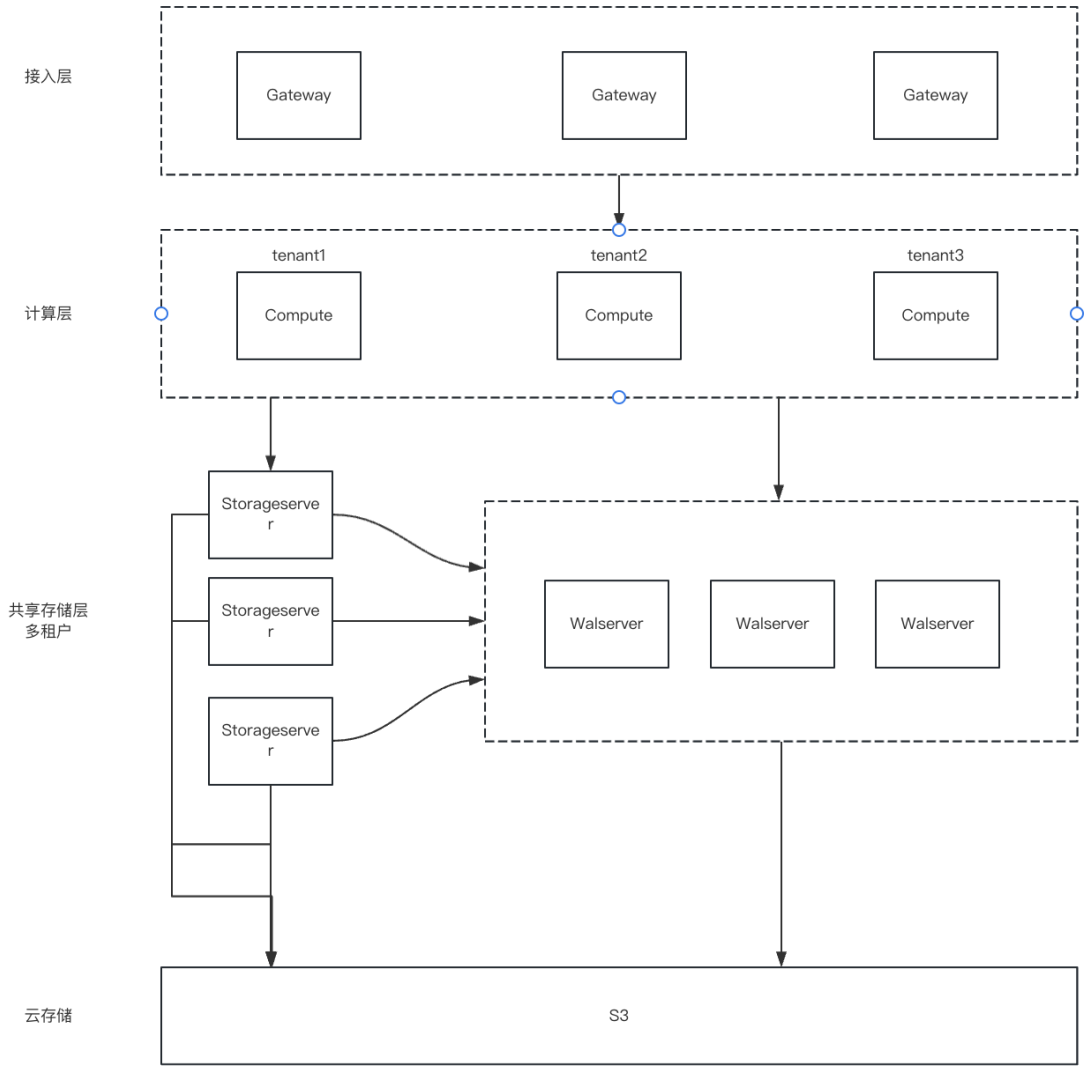
使用 codis 或者 twemproxy 构建数据分片



Serverless

要成为 serverless 数据库，需要在当前架构的基础上做改动，具备多租户、存算分离、云存储等功能，针对当前 pika 两种使用模式，也相应的设计了两种方案

简易方案



gateway：接受用户请求，支持 redis 协议，并转发到用户节点

computer：计算节点，每个租户有自己单独的计算节点，可弹性扩所容

storageserver：存储节点，从 walserver 同步日志，并定时同步数据到云存储，同时接受计算节点的查询请求

walserver：wal 日志节点，三副本保持高可用，可将 wal 日志写入本地磁盘或者云存储

object storage：云存储

多租户

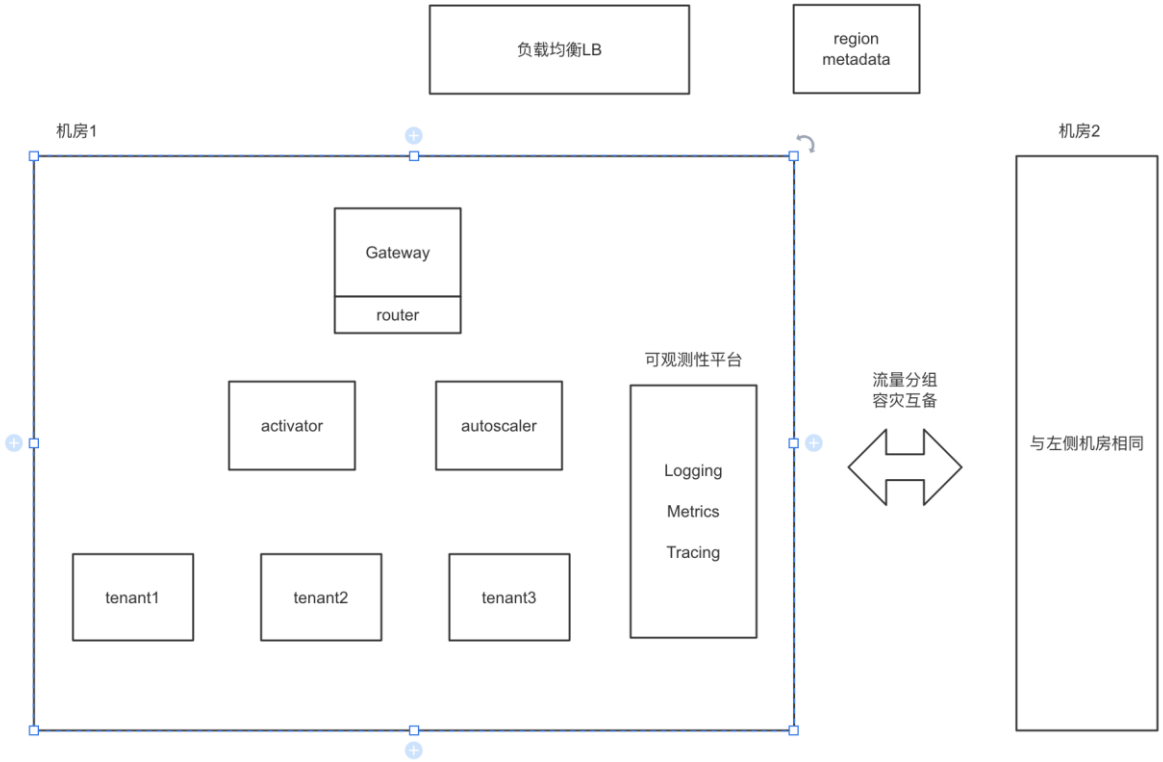
目前 serverless 模式的数据库都是在公有云平台上创建账号，平台会将租户信息放在数据库连接串中

```
    pika://alex:AbC123dEf@ep-cool-darkness-123456.us-east-2.aws.neon.tech/dbname
^           ^                                  ^
      |- role                        |- hostname                    |- database
```

存算分离

计算层

整体结构如下所示，可以细分为两大功能，服务路由以及弹性伸缩



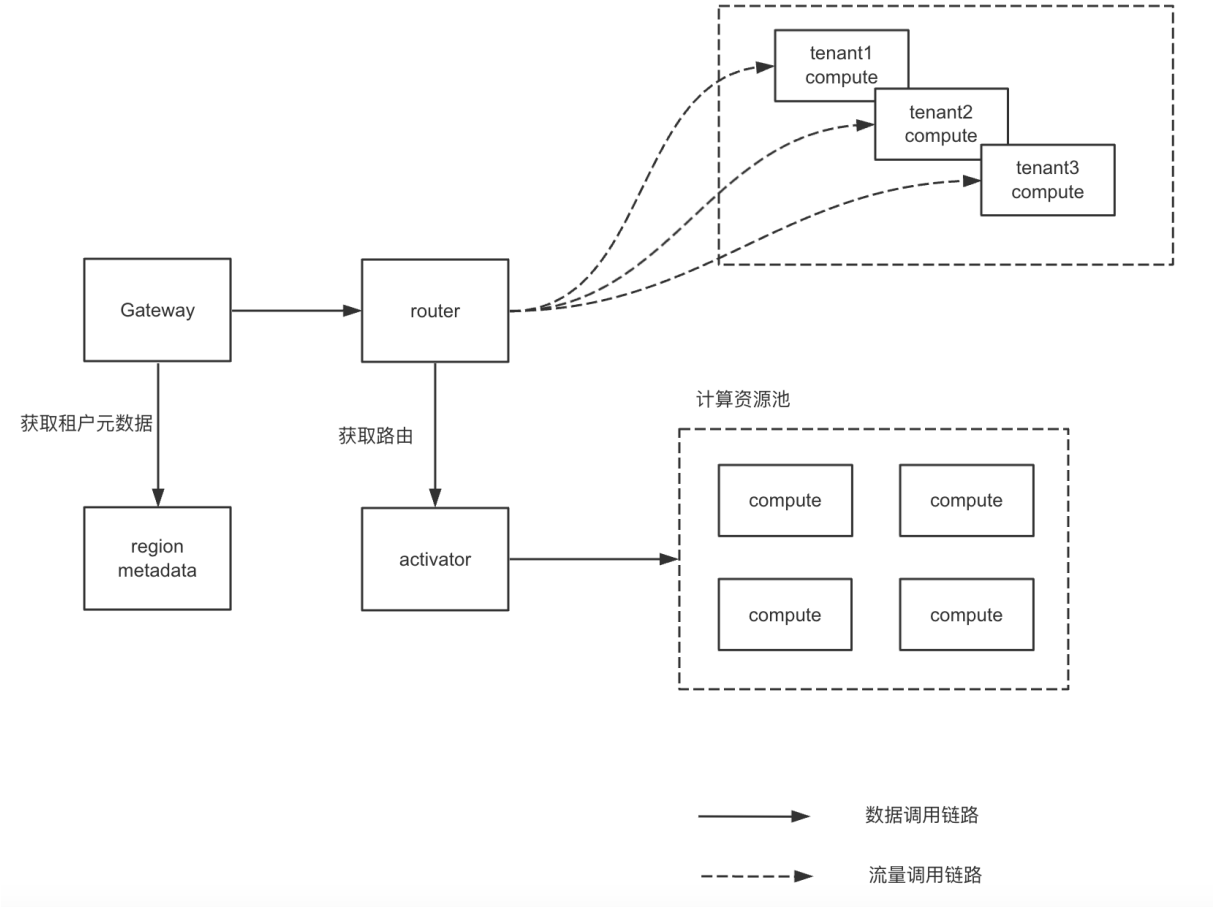
region metadata

存储租户与机房的元数据信息，多机房架构时需要用到，有可能租户的计算节点不在本机房内，需要存储租户与机房的元数据信息，目前可设计为简单的映射关系，后续可添加其他属性

租户	属性
tenant1	{gateway: "192.168.1.1"}

gateway

gateway 主要做入口网关，识别 redis 协议并做路由转发，基本结构如下所示

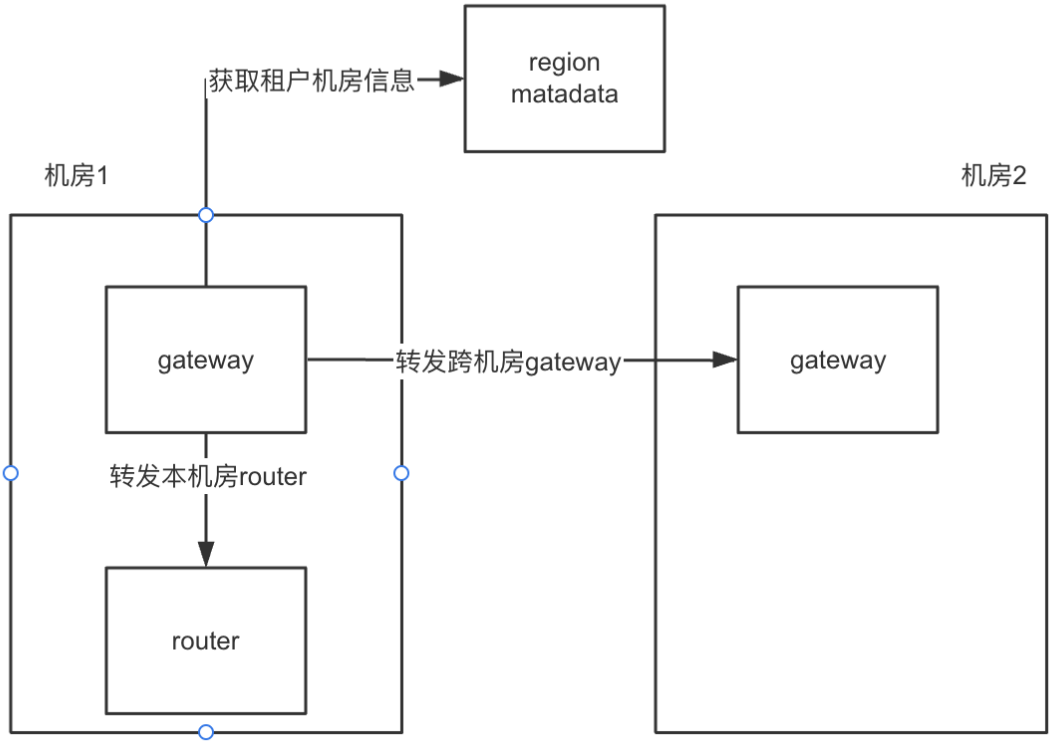


1）协议识别

需要支持 redis 协议

2）路由

路由是 gateway 的主要功能，主要包括夸机房路由和版本路由，路由逻辑如下所示

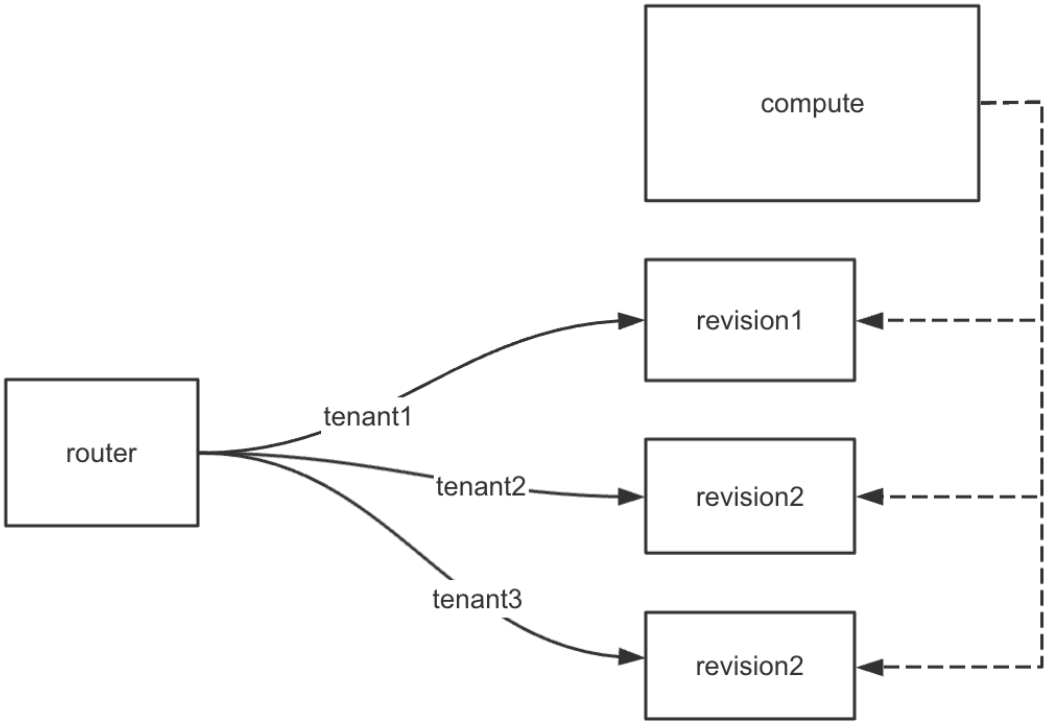


跨机房路由

从 region metadata 中查询租户的机房信息，如果不在本机房内，则转发到对应的机房，否则转发到 router 模块，router 是 gateway 内置的模块，不是单独的组件

版本路由

计算节点会有存在多个版本的情况，版本升级会出现多个版本的计算池，当用户使用时，需要合理的为用户分配一种版本的计算节点，也方便我们使用灰度、蓝绿等策略做版本升级

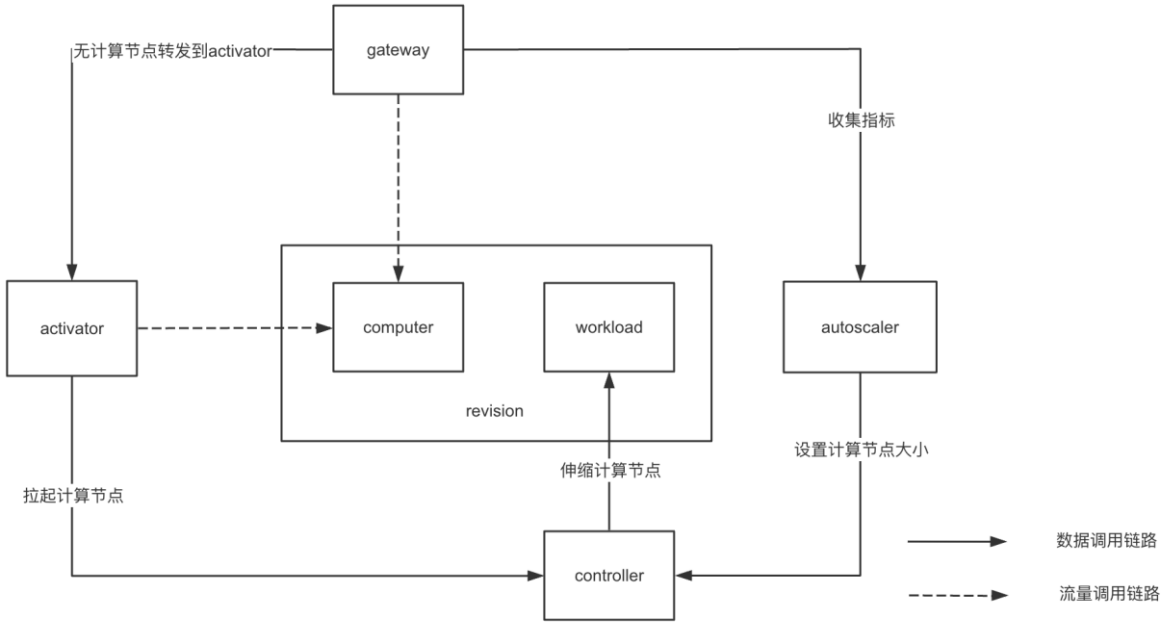


其中 revision 表示计算节点的版本，比如 v1，v2，可以通过一定策略将用户路由到不同版本，可支持如下策略

策略	示例
按比例路由	5%路由到 v1，95%路由到 v2
按地域路由	华东的确路由到 v1，其他地方路由到 v2
按租户路由	租户 1 开头的路由到 v1，其他路由到 v2

activator

activator 是一个激活器，用于解决冷启动问题，当用户无计算节点时，请求会被转发到 activator。activator 会缓存用户请求，并拉起计算节点，计算节点启动成功后，activator 再转发请求到计算节点



pool

这里是计算节点池，为了解决冷启动问题，当用户计算节点每次被拉起时，如果重新创建会很耗时，所以池化计算节点，可以实现快速启动，启动时直接从池中获取计算节点即可。计算池的大小可设置为恒定的数目，也可根据历史水位动态设置大小

controller

controller 用于控制用户计算节点数目，接收 activator 和 autoscaler 的请求，设置计算节点数目

workload

workload 是用于控制用户计算节点的上层工作负载，主要包括以下功能

1）从计算池中获取计算节点作为用户自身使用

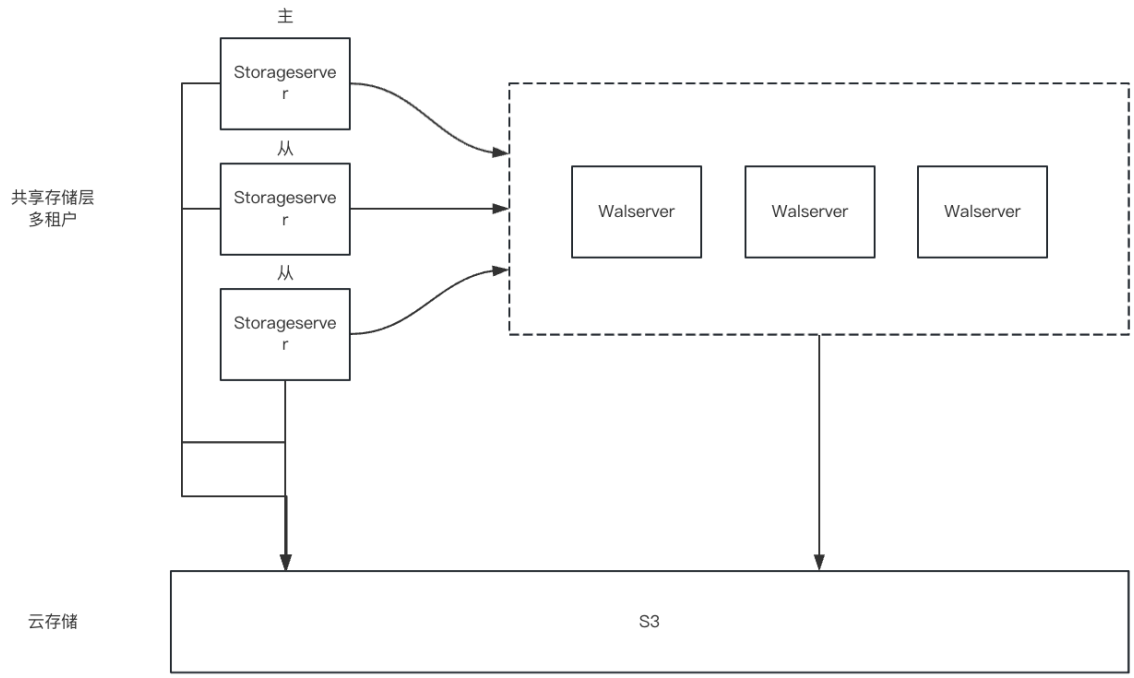
2）扩所容用户节点，并能够所容指定的用户节点

autoscale

负责租户计算节点的弹性伸缩，可以伸缩到 0，真正的按需收费，从 gateway 中获取连接请求的活跃度，如果一段时间内无读写请求，可销毁当前计算节点，当连接数超过一定数目时，或者 cpu、内存负载较高时，可扩容节点

指标	解释
计算节点的 cpu、内存	cpu、内存大于阈值时扩容，小于阈值时所容，但不能缩容到 0
计算节点租户连接活跃度	当计算节点上连接不活跃，可缩容当前节点

存储层



walserver

持久地存储 WAL，直到它被 storageserver 处理并上传到云存储。同时 WAL 也可以在对象存储中持久化保存。如果 Safekeeper 可以把数据存储在高性能 SSD 上，那么数据修改可以快速落盘，实现较大并发的数据写入

三副本，使用 raft 协议保存一致性，并将数据存储在 SSD 上，可保证快速写入

storageserver

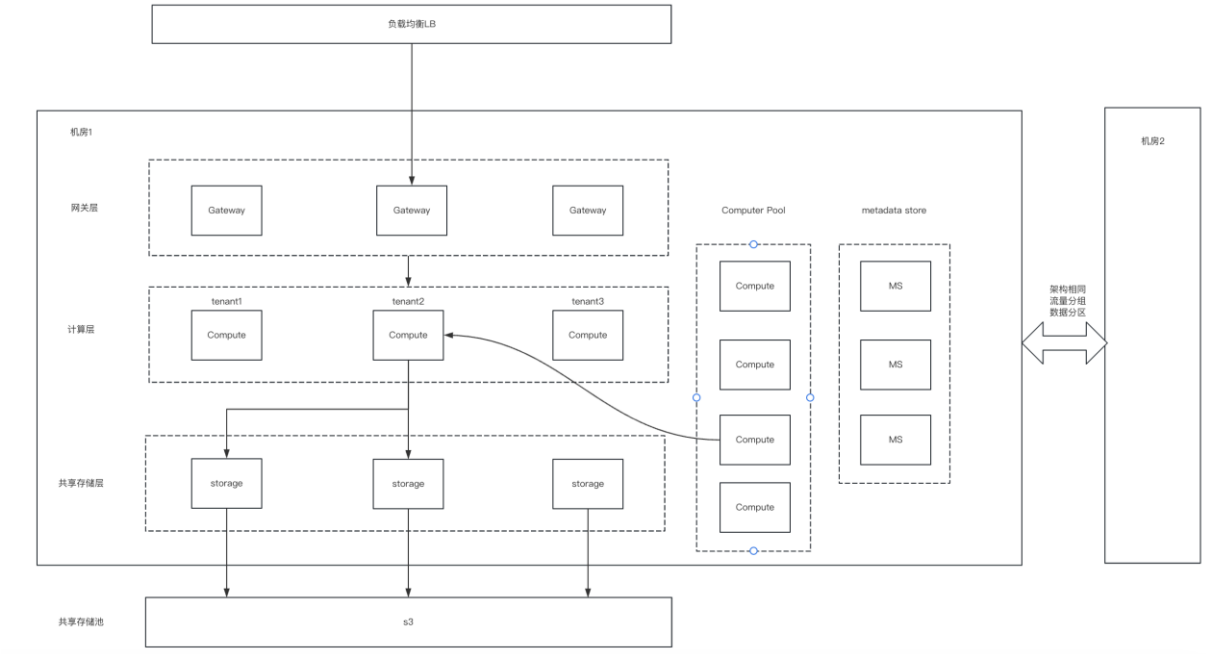
负责处理读取请求。为此， storageserver 将传入的 WAL 流处理为自定义存储格式，使所有 storage 版本都易于访问。storageserver 还将数据上传到云对象存储，并按需下载数据。同时，storageserver 还承担了一个缓冲层，存储了经常会被数据库访问的较热的数据

目前 storageserver 还是主从架构，主写从读，都从 walserver 拉去 wal 日志，从节点可以横向扩展。如果需要采用分布式分片架构，推荐使用复杂模式

walserver 和 storageserver 都是多租户的，数据也是按照多租户模式来存储的，如下所示

```
<working directory>
├── redo.log
├── undo.pid
└── tenants
    ├── 537cffa58a4fa557e49e19951b5a9d6b
    ├── de182bc61fb11a5a6b390a8aed3a804a
    └── ee6016ec31116c1b7c33dfdfca38891f
```

复杂模式



复杂模式下多租户的实现与简易模式相同，存算分离会有些区别

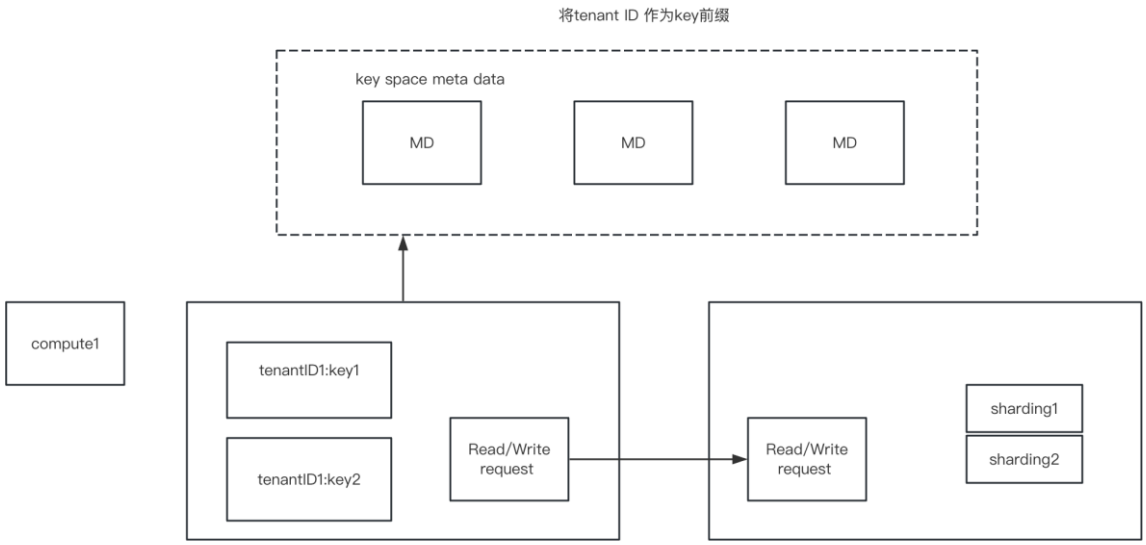
存算分离

计算层

计算层与简易模式下相同

存储层

复杂模式下存储节点不在是多租户模式，将租户作为 key 的前缀，对 key 进行了分片后存储，实现数据的分布式存储



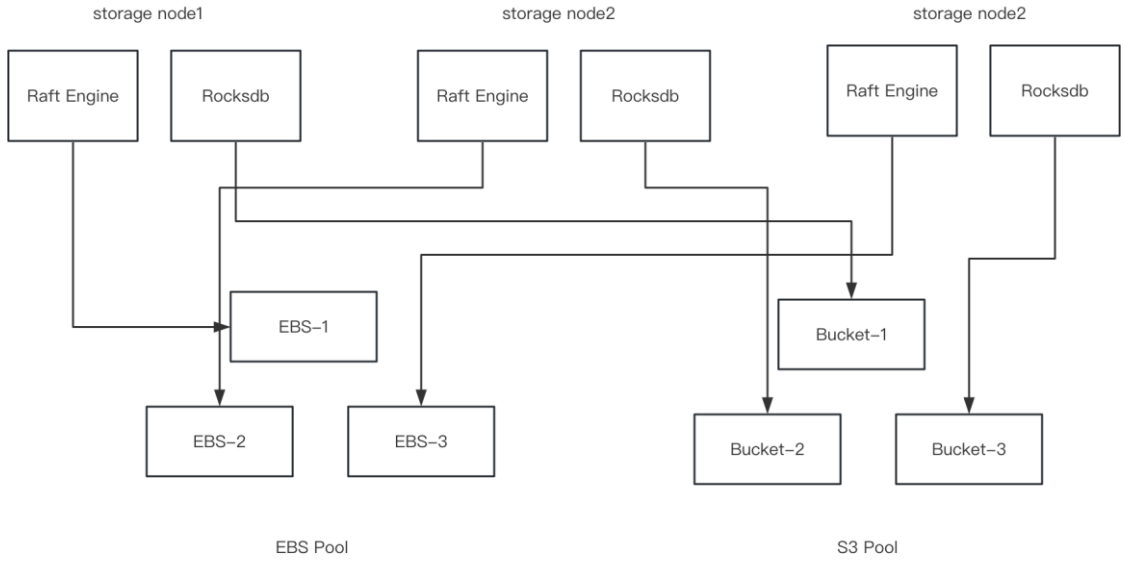
MD

元数据中心，存储分片信息，分片与实际存储节点的元数据信息

computer

此时计算节点需要从将 tenantID 与 key 拼接成新的 key，并从元数据中心获取分片对应的存储节点信息，将数据写入存储节点信息

stroage



存储节点中原有的 raft 协议写三副本 wal 日志现在只需要有单个节点写入块存储中即可，由云存储保证高可用，块存储能够保证快速写入。rocksdb 将数据写入到 s3 对象存储中，由 s3 保证数据的高可用。

reference

https://www.modb.pro/db/77371

https://github.com/OpenAtomFoundation/pika/wiki/Support-Cluster-Slots

https://neon.tech/docs/connect/connect-from-any-app
https://blog.csdn.net/devcloud/article/details/118360126
https://tidb.net/book/tidb-monthly/2023/2023-02/feature-indepth/tidb-serverless-and-technology-ecology-overview
https://www.infoq.cn/article/hyi6km1ebkoeswfdvty6
https://www.infoq.cn/article/az7tqvww0sabttuoupj5
https://www.51cto.com/article/757275.html
https://tech.meituan.com/2021/04/21/nest-serverless.html
https://www.sofastack.tech/blog/knative-serverless-kubecon-na2019/