ZStack 技术白皮书精选

ZStack--查询 API

扫一扫二维码，获取更多技术干货吧

版权声明

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple （简单）、Strong （健壮）、Smart （智能）、Scalable （弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

ZSTACK--查询 API

IaaS 软件用户面临的共同挑战是如何快速、准确地找到一个想要的资源；例如，从 10,000 台虚拟机中发现有 EIP（16.16.16.16）的虚拟机。大多数 IaaS 软件通过 API 中的特定查询逻辑解决这个问题。ZStack 不用特定查询，而是配备了一个框架，这个框架可以自动为每个资源的每个字段生成查询，并联合跨越了多个资源的查询，帮助用户管理云端数量庞大的资源。

动机

一个中型的云可以管理几百台物理主机和成千上万台虚拟机，因为 IaaS 软件很少有全部的查询 API，导致寻找想要的资源成为挑战。大多数 IaaS 软件只允许用户使用少量条件（如 name，UUID）查询资源，这些条件硬编码在查询 API 中。如果用户想要使用硬编码之外的条件做一个查询，例如，通过创建日期查询虚拟机，他们可能不得不最终列出所有虚拟机，然后用 `for..loop` 来过滤结果。使用任意字段查询资源至今在大多数 IaaS 软件都不被完全支持，更不用说联合查询；例如，如果用户想要找到一个虚拟机，这个虚拟机的网卡应用了特定的安全组规则，他们可能不得不列出所有资源（虚拟机，安全组），然后做两次 `for..loop`。

另一方面，相比类似 JIRA 的软件，大多数 IaaS 软件的 UI 是最粗糙简陋的。许多开发人员可能并没有意识到糟糕 UI 的根源不是因为 UI 开发人员缺乏 CSS/HTML/JavaScript 的技能，而是软件本身并不能提供强大的 API 来支持复杂的 UI；例如，为了实现一个类似 JIRA 过滤器的功能，即只显示满足指定条件的资源，UI 可能需要做很多的需要 `listing all then filtering by for..loop` 的后置处理工作，这些后置处理工作将把大量的资源拧在一起。

IaaS 软件因此饱受折磨了一段时间；对此的解药就是要提供一种机制，这种机制可以自动为每个资源的每个字段都生成查询，而且可以处理 join 查询。

问题

大多数 IaaS 软件使用关系型数据库（如 MySQL）作为后台数据库，在这种数据库中资

源通常被安排在单独的表中，比如虚拟机表，主机表，云盘表。对于每一个资源，都有一个 API 用来获取该资源的单独的一条，它可能被命名为 `describe API`、`list API` 或 `query API`；这些 API 通常有硬编码的参数，用来暴露一部分数据库表的列，允许用户通过少量的查询条件查询资源；这些参数是精心选择的，通常是对 API 设计者自身非常重要的列，例如，name、UUID。然而，由于并不是所有的列都被暴露，用户经常遇到他们想查询的列不存在的情况，这样他们就必须检索所有的资源，然后使用一个或多个 `for..loop` 进行后置处理。

一个复杂的查询场景，可能需要使用联合查询，这种查询通常跨越多个数据库表；例如，找到一个 EIP 为 16.16.16.16 的虚拟机，它可能涉及虚拟表、网卡表和 EIP 表。一些 IaaS 软件使用数据库视图解决这个问题，这是另一种硬编码方式，只能以固定的格式 join 选中的表，而在现实中表是能以非常复杂的方式被 join 的。在软件升级过程中，如果一个视图指向的任一表已经改变了的话，视图也需要进行数据库迁移操作。

查询 API

为了避免在 API 中手动编码查询逻辑，并给用户提供能在任何地方查询任何东西的灵活的查询，ZStack 创建了一个框架，这个框架可以自动为所有资源生成查询，并且不需要开发者写代码去实现查询逻辑；更进一步，该框架还可以生成各种 join 查询，只要所需的表已经通过外键连接。

在以下篇幅中，我们将用 zone 作为一个例子来阐述这个令人惊叹的框架。一个 zone 在数据库中有下面的这些列：

FIELD	DESCRIPTION
uuid	zone UUID
name	zone name

description	zone description
state	zone state
type	zone type
createDate	the time the zone was created
lastOpDate	the last time the zone was operated

用户可以通过任何一个字段或字段组合来查询 zone，并采用常规的 SQL 比较运算符如'=', '!=', '>', '>=', '<', '<=', 'in', 'not in', 'is null', 'is not null', 'like', 'not like'。

注意：在命令行工具中，一些运算符有不同的格式：'in'(?=), 'not in'(!?=), 'is null'(=null), 'is not null'(!=null), 'like'(~=), 'not like'(!~=).

```
QueryZone name=west-coast-zone

QueryZone name=west-coast-zone state=Enabled
```

因为 zone 是 ZStack 中主要资源的祖先，很多资源都或多或少和它有关系；例如，一个运行中的虚拟机总是在一个 zone 内。像这种关系可以生成联合查询，如：

```
QueryZone vmInstance.name=web-vm1
```

如上表格所示，一个 zone 不会暴露任何叫 `vmInstance` 的字段，但在上述查询中有一个条件是由'vmInstance'开始的。这种查询在 ZStack 中称为*扩展查询*。这里 `vmInstance` 代表 VM 表，VM 表有一个字段为 `zoneUuid`（外键）指向 zone 表，因此查询框架可以理解它们的关系并生成联合查询。上面的例子可以被解释为"*寻找运行着名字为web-vm1 的虚拟机的zone*"。进一步扩展这个例子，因为虚拟机网卡表有外键指向 VM 表，并且 EIP 表有外键指向虚拟机

网卡表，查询 zone 也可以使用 EIP 作为条件：

```
QueryZone vmInstance.vmNics.eip.vipIp=16.16.16.16
```

查询被解释为*“查找一个区域，它上面的虚拟机的网卡的 EIP 为 16.16.16.16”*。现在您知道了查询接口的强大之处了！我们甚至可以创建一些非常复杂的查询：

```
QueryVolumeSnapshot
volume.vmInstance.vmNics.l3Network.l2Network.attachedClusterUuids=13238c8e0591444e9
160df4d3636be82
```

这个复杂的查询目的是找到磁盘快照，目标磁盘快照是由虚拟机磁盘创建的，而该虚拟机有网卡在 L3 网络上，这个 L3 网络的父 L2 网络则是附加在一个集群上的，这个集群的 uuid 是 13238c8e0591444e9160df4d3636be82。不要惊慌，你很少需要这么复杂的查询，但它确实证明了框架的能力。此外，SQL 的一些特性例如选择字段、排序、计数和分页也是支持的：

```
QueryL3Network name=L3-SYSTEM-PUBLIC count=true

QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec limit=10

QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec start=10 limit=100

QueryL3Network fields=name,uuid l2NetworkUuid=33107835aee84c449ac04c9622892dec

QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec sortBy=createDate
sortDirection=desc
```

实现

尽管查询 API 功能是如此强大，实现却是非常简洁的。当添加一个新的资源时，开发人员不需要写任何关于查询逻辑的代码，除了定义查询 API 和资源本身。要实现 zone 的查询 API，开发人员需要：

1. 使用查询元数据注解 zone 的 inventory

```
@Inventory(mappingVOClass = ZoneVO.class)

@PythonClassInventory

@ExpandedQueries({

        @ExpandedQuery(expandedField = "vmInstance", inventoryClass = VmInstanceInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "cluster", inventoryClass = ClusterInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "host", inventoryClass = HostInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "primaryStorage", inventoryClass =

PrimaryStorageInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "l2Network", inventoryClass = L2NetworkInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "l3Network", inventoryClass = L3NetworkInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid"),

        @ExpandedQuery(expandedField = "backupStorageRef", inventoryClass =

BackupStorageZoneRefInventory.class,

                foreignKey = "uuid", expandedInventoryKey = "zoneUuid", hidden = true),

      })

@ExpandedQueryAliases({
```

```
       @ExpandedQueryAlias(alias = "backupStorage", expandedField =
"backupStorageRef.backupStorage")

      })

     public class ZoneInventory implements Serializable{

    private String uuid;

    private String name;

    private String description;

    private String state;

    private String type;

  private Timestamp createDate;

  private Timestamp lastOpDate;

      }
```

上面的注解声明了 zone 和其他资源之间的关系，这是 zone 扩展查询的基础。

2.定义一个查询 API

```
@AutoQuery(replyClass = APIQueryZoneReply.class, inventoryClass = ZoneInventory.class)

     public class APIQueryZoneMsg extends APIQueryMessage {

      }
```

3.在区域的 API 配置文件中声明查询 API

```
<?xml version="1.0" encoding="UTF-8"?>

      <service xmlns="http://zstack.org/schema/zstack">

    <id>zone</id>
```

```
    <interceptor>ZoneApiInterceptor</interceptor>



    <message>

        <name>org.zstack.header.zone.APIQueryZoneMsg</name>

        <serviceId>query</serviceId>

    </message>

     </service>
```

API `APIQueryZoneMsg` 通过指定服务 ID `query` 被路由到查询服务。就是这样了，查询逻辑不需要一行代码；查询服务会把其余部分自动完成。所有的 ZStack 查询 API 都像这样定义，添加新资源的查询 API 是非常容易的。

当前限制

主要的限制是在查询条件中，只有逻辑 `AND` 是被支持的，`OR` 是不被支持的。例如：

```
QueryZone name=west-coast-zone state=Enabled
```

上述查询语句可以被解释为“寻找区域名字为 west-coast 且 state 是 Enabled 的区域”我们这么做的原因是我们由 ZStack 源代码中 SQL 的使用分析得出 99%的组合的查询条件都是基于 `AND` 逻辑的。另一方面，如果逻辑 `OR` 在没有创建 DSL 的情况下就被引入，要保持代码简洁是非常困难的。然而，在很多情况下，`OR` 可以使用比较操作 `in(?=)`实现：

```
QueryZone name=west-coast-zone state?=Enabled,Disabled
```

上述例子表述的是“寻找名字为 west-coast 的区域，并且它的状态是 Enabled 或 Disabled”，将来，我们将引入 DSL 风格的查询语言，例如：

```
QueryZone name=west-coast-zone AND (state=Enabled OR state=Disabled)
```

总结

这篇文章中，我们演示了 ZStack 的查询 API。通过使用这个强大的工具，用户能以类似关系型数据库的方式查询任何资源。将来，ZStack 将建立一套高级的 UI，它可以使用查询 API 创建各种各样的视图（过滤器），例如，展示所有运行在同一 L3 网络的虚拟机，为 IaaS UI 的用户体验带来革命性的改变。