第 11 章 用户管理

Proxmox VE 支持多种身份认证方式，例如 Linux PAM，内部 Proxmox VE 认证服务，微软活动目录。

基于角色的用户和权限管理覆盖了所有对象（虚拟机，存储，节点等），能够实现细粒度的访问控制。

11.1 用户

Proxmox VE 的用户信息保存在/etc/pve/user.cfg 中。但该文件中不保存口令信息，用户通过本章后续介绍的认证域进行认证。因此，Proxmox VE 需要联合用户名和认证域信息<userid>@<realm>才可以定义完整的用户身份。

配置文件中每条用户记录同时包含了以下信息

- 名
- 姓
- 电子邮件
- 组
- 可选的过期时间
- 用户信息注释
- 用户启用/禁用标志
- 双因子认证密钥

11.1.1 系统管理员

系统 root 用户可以通过 Linux PAM 域登录系统，并拥有最高管理权限。该用户不能被删除，但其属性可以被修改，系统邮件会发送到为该用户分配的电子邮件地址。

11.1.2 组

用户可以同时加入多个组。组可以有效简化访问权限控制工作。以组为单位赋予访问权限比直接向单个用户赋权要方便的多，最终得到的访问控制列表也要短的多，便于处理。

11.2 认证域

Proxmox VE 用户实际上其他外部认证域用户的一个副本。认证域信息都保存在 /etc/pve/domains.cfg。以下是可用的认证域：

Linux PAM 标准认证

该认证域要求在集群的所有节点上创建相同的 Linux 系统用户（例如，使用 adduser 命令创建用户），并使用 Linux 口令认证用户身份。

useradd heinz

passwd heinz

groupadd watchman

usermod -a -G watchman heinz

Proxmox VE 认证服务器

用户口令保存在 Unix 风格的口令文件（/etc/pve/priv/shadow.cfg）中。口令使用 SHA-256 哈希算法加密。该方式是小规模（或中等规模）环境下最便于使用的认证方式。用户在 Proxmox VE 内部即可完成身份认证，无须任何外部支持，所有用户身份都由 Proxmox VE 管理，并可在 WebGUI 界面直接修改口令。

LDAP

也可以使用 LDAP 服务器（例如 openldap）进行用户身份认证。LDAP 支持部署备用服务器，并允许通过加密的 SSL 连接传递认证信息。

LDAP 将在基本域名（base_dn）下搜索用户属性名（user_attr）指定的用户名。

例如，如果一个用户的 ldif 身份信息记录如下：

```
# user1 of People at ldap-test.com
dn: uid=user1,ou=People,dc=ldap-test,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: user1
cn: Test User 1
sn: Testers
description: This is the first test user.
```

则基本域名为“ou=People,dc=ldap-test,dc=com”，用户属性为“uid”。

如果 Proxmox VE 在请求验证用户身份前需要先认证（bind）ldap 服务器的真实性，可以通过/etc/pve/domains.cfg 中的 bind_dn 属性配置认证域名称。认证口令则保存在/etc/pve/priv/ldap/<realmname>.pw（例如 /etc/pve/priv/ldap/my-ldap.pw），其中仅有一行明文口令信息。

微软活动目录

该方式需要指定认证域和认证服务器。类似 ldap，可以配置备用服务器，可选端口和 SSL 加密。

11.3 双因子认证

以上每种认证方式都可以进一步结合双因子认证以提高认证强度。只需在添加或修改认证域时从 TFA 下拉框中选择可用的双因子认证方法即可。当一种认证域启用双因子认证后，只有能提供双因子认证信息的用户才可以登录。

目前有两种可用的双因子认证方法：

时间令牌（TOTP）

该方式采用标准 HMAC-SHA1 算法，用当前时间和用户密钥计算得到的哈希值认证用户身份。而时间步进长度和口令长度都是可以配置的。

一个用户可以设置多个密钥（用空格分隔开），可用 Base32（RFC3548）或 16 进制形式记录。

Proxmox VE 提供了密钥生成工具（oathkeygen），可以产生 Base32 格式的随机密钥，与多种 OTP 工具直接配合使用，例如 oathtool 命令行工具，Google 认证器或 Android 应用 FreeOTP。

YubiKey 令牌

使用 YubiKey 进行双因子认证，必须预先配置 Yubico API ID，API 密钥和可用的服务器 URL，同时用户必须拥有 YubiKey 硬件。如果要从 YubiKey 提取 Key ID，需要将 YubiKey 插入计算机 USB 接口并激活，然后将输入口令的前 12 个字符拷贝到用户的 Key ID 字段。

关于 YubiCloud 使用方法和如何建立自己的认证服务器，可查看 YubiKey OTP 文档。

11.4 权限管理

用户进行任何操作前（例如查看、修改、删除虚拟机配置），都必须被赋予合适的权限。

Proxmox VE 采用了基于角色和对象路径的权限管理系统。权限管理表中的一个条目记录了用户或组在访问某个对象或路径时所拥有的角色。也就是说，每条访问策略都可以用（路径，用户，角色）或（路径，组，角色）三元组来表示，其中角色包含了允许进行的操作，路径标明了操作的对象。

11.4.1 角色

角色实际上是一个权限列表。Proxmox VE 预定义了多个角色，能够满足大部分的管理需要。

- Administrator：拥有所有权限

- NoAccess：没有任何权限（用于禁止访问）
- PVEAdmin：有权进行大部分操作，但无权修改系统设置（Sys.PowerMgmt，Sys.Modify，Realm.Allocate）。
- PVEAuditor：只有只读权限。
- PVEDatastoreAdmin：创建和分配备份空间和模板。
- PVEDatastoreUser：分配备份空间，查看存储服务。
- PVEPoolAdmin: 分配资源池。
- PVESysAdmin: 分配用户访问权限，审计，访问系统控制台和系统日志。
- PVETemplateUser: 查看和克隆模板。
- PVEUserAdmin: 用户管理。
- PVEVMAdmin: 管理虚拟机。
- PVEVMUser: 查看，备份，配置 CDROM，访问虚拟机控制台，虚拟机电源管理。

在 WebGUI 可以查看系统预定义的所有角色。

目前，新增角色只能通过命令行进行，如下：

pveum roleadd PVE_Power-only -privs "VM.PowerMgmt VM.Console"

pveum roleadd Sys_Power-only -privs "Sys.PowerMgmt Sys.Console"

11.4.2 权限

权限是指进行某种操作的权力。为简化管理，一组权限可以被编组构成一个角色，角色可以用于制定权限管理表的条目。注意，权限不能被直接赋予用户和对象路径，而必须借助角色才可以。

目前有如下权限：

节点/系统相关的权限

- Permissions.Modify：修改访问权限
- Sys.PowerMgmt：管理节点电源（启动，停止，重启，关机）
- Sys.Console：访问节点控制台

- Sys.Syslog：查看 syslog
- Sys.Audit：查看节点状态/配置
- Sys.Modify：创建/删除/修改节点网络配置参数
- Group.Allocate：创建/删除/修改组
- Pool.Allocate：创建/删除/修改资源池
- Realm.Allocate：创建/删除/修改认证域
- Realm.AllocateUser：将用户分配到认证域
- User.Modify：创建/删除/修改用户访问权限和详细信息

虚拟机相关的权限

- VM.Allocate：创建/删除虚拟机
- VM.Migrate：迁移虚拟机到其他节点
- VM.PowerMgmt：电源管理（启动，停止，重启，关机）
- VM.Console：访问虚拟机控制台
- VM.Monitor：访问虚拟机监视器（kvm）
- VM.Backup：备份/恢复虚拟机
- VM.Audit：查看虚拟机配置
- VM.Clone：克隆/复制虚拟机
- VM.Config.Disk：添加/修改/删除虚拟硬盘
- VM.Config.CDROM：弹出/更换 CDROM
- VM.Config.CPU：修改 CPU 配置
- VM.Config.Memory：修改内存配置
- VM.Config.Network：添加/修改/删除虚拟网卡
- VM.Config.HWType：修改模拟硬件类型
- VM.Config.Options：修改虚拟机的其他配置

- VM.Snapshot：创建/删除虚拟机快照

存储相关的权限

- Datastore.Allocate：创建/删除/修改存储服务，删除存储卷
- Datastore.AllocateSpace：在存储服务上分配空间
- Datastore.AllocateTemplate：分配/上传模板和 iso 镜像
- Datastore.Audit：查看/浏览存储服务

11.4.3 对象和路径

访问权限是针对对象而分配的，例如虚拟机，存储服务或资源池。我们采用了类似文件系统路径的方式来标识这些对象。所有的路径构成树状结构，用户可以选择将高层对象获得的权限（短路径）扩展到下层的对象。

路径是可模板化的。当 API 调用申请访问一个模板化路径时，路径中可以包含 API 调用的参数。其中 API 参数需要用花括号括起来。某些参数会被隐式地从 API 调用的 URI 中获取。例如在调用/nodes/mynode/status 时，路径/nodes/{node}实际上申请了/nodes/mynode 的访问权限。而在对路径/access/acl 的 PUT 请求中，{path}实际上引用了该方法的 path 参数。

一些示例如下：

- /nodes/{node}：访问 Proxmox VE 服务器
- /vms：所有的虚拟机
- /vms/{vmid}：访问指定虚拟机
- /storage/{storeid}：访问指定存储服务
- /pool/{poolname}：访问指定存储池中虚拟机
- /access/groups：组管理操作
- /access/realms/{realmid}：管理指定认证域

权限继承

如前所述，对象路径全体构成了类似文件系统的树状结构，而权限能够自上而下地继承下去（继承标识默认是启用的）。我们采用了以下的继承策略：

- 针对单一用户的权限将覆盖针对组的权限。
- 针对组赋权后，组内所有用户自动获得赋限。
- 明确的赋权会覆盖从高层继承来的赋权。

11.4.4 资源池

资源池主要用来将虚拟机和存储服务组织起来，并形成一个组。当对资源池赋予访问权限后（/pool/{poolid}），其中所有成员都会继承该权限，从而大大简化访问控制配置工作。

11.4.5 我究竟需要哪些权限？

在 http://pve.proxmox.com/pve-docs/api-viewer/记录了每一个方法所需的 API 调用权限。

所需的权限以列表形式表示，可以看作一个由访问权限检查函数构成的逻辑树。

["and", <subtests>...] and ["or", <subtests>...]

当前列表中的所有（and）或任意一个（or）权限需要被满足。

["perm", <path>, [<privileges>...], <options>...]

该路径是一个模板参数（查看对象和路径一节）。访问目标路径时，列表中所有的（或任意一个，如果使用了 any 选项）权限需要被满足。如果指定了 require-param 选项，则需要满足指定的参数权限，除非 API 调用时标明该参数为可选的。

["userid-group", [<privileges>...], <options>...]

调用方需要拥有/access/groups 所列出的任意权限。此外，根据是否设置 groups_param 参数，还需要额外进行两个权限检查：

- 设置了 groups_param：API 调用使用了不可选的组参数，调用方必须对参数引用的所有组拥有该组所列出的任意权限。

- 未设置 groups_param：通过 userid 参数传递的用户必须存在，并且是组的成员，而调用方拥有所列出的任意权限（通过/access/groups/<group>路径）。

["userid-param", "self"]

向 API 传递的 userid 参数值必须和申请进行操作的用户一致。（通常和 or 联合使用，以允许用户在没有权限的情况下在自身执行操作。）

["userid-param", "Realm.AllocateUser"]

用户需要对/access/realm/<realm>,拥有 Realm.AllocateUser 访问权。其中<realm>是用户通过 userid 参数传递的认证域。注意，用户不一定需要真的存在，因为用户 ID 是以<username>@<realm>形式传递的。

["perm-modify", <path>]

其中 path 是一个模板化参数（参见对象和路径一节）。用户需要拥有 Permissions.Modify 权限，或根据以下不同路径拥有相应权限：

- /storage/...：需要额外拥有权限’ Datastore.Allocate`
- /vms/...：需要额外拥有权限’ VM.Allocate`
- /pool/...：需要额外拥有权限’ Pool.Allocate`

如果路径为空，需要对/access 拥有 Permission.Modify 权限。

11.5 命令行工具

大部分用户使用 WebGUI 就能够完成用户管理任务了。但 Proxmox VE 还提供了一个全功能的命令行工具 pveum（ “Proxmox VE User Manager”的缩写）。由于 Proxmox VE 的命令行工具都通过封装 API 实现的，因此你也可以通过调用 REST API 来使用这些功能。

如下是一些使用示例。如需要显示帮助信息，可运行：

pveum

或（针对特定命令显示更详细的信息）

pveum help useradd

创建新用户：

pveum useradd testuser@pve -comment "Just a test"

设置或修改口令（不是所有认证域都支持该命令）：

pveum passwd testuser@pve

禁用用户：

pveum usermod testuser@pve -enable 0

创建新用户组：

pveum groupadd testgroup

创建新角色：

pveum roleadd PVE_Power-only -privs "VM.PowerMgmt VM.Console"

11.6 实际应用示例

11.6.1 管理员组

一个很实用的特性是创建一组具有全部管理权限的管理员用户（不使用 root 用户）。

定义管理员组：

pveum groupadd admin -comment "System Administrators"

赋予权限：

pveum aclmod / -group admin -role Administrator

向管理员组添加管理员用户：

pveum usermod testuser@pve -group admin

11.6.2 审计员

赋予用户或用户组 PVEAuditor 角色就可以赋予相应用户对系统的只读权限。

例 1：允许用户 joe@pve 查看系统所有对象

pveum aclmod / -user joe@pve -role PVEAuditor

例 2：允许用户 joe@pve 查看所有虚拟机

pveum aclmod /vms -user joe@pve -role PVEAuditor

11.6.3 分配用户管理权限

如果需要将用户管理权限赋予 joe@pve，可以运行如下命令：

pveum aclmod /access -user joe@pve -role PVEUserAdmin

之后，joe@pve 用户就可以添加和删除用户，修改其他用户的口令和属性。这是一个权限非常大的角色。你应该将该权限限制在指定的认证域和用户组。以下是限制 joe@pve 仅能修改 pve 认证域中 customers 用户组用户的示例：

pveum aclmod /access/realm/pve -user joe@pve -role PVEUserAdmin

pveum aclmod /access/groups/customers -user joe@pve -role PVEUserAdmin

---

- 注意

执行以上命令后，joe@pve 用户能够添加用户，但添加的用户只能属于 pve 认证域中的 customers 用户组。

11.6.4 资源池

一个企业往往设立有多个部门，将资源和管理权限分配给各个部门是很常见的做法。资源池是一组虚拟机和存储服务的集合，你可以在 WebGUI 创建资源池，然后向资源池添加资源（虚拟机，存储服务）。

你可以向资源池赋予访问权限，这些权限会被其成员自动继承获取。

假定你有一个软件开发部，首先创建用户组

pveum groupadd developers -comment "Our software developers"

然后为该组创建一个新用户

pveum useradd developer1@pve -group developers –password

- 注意

参数-password 将会提示你设立用户口令。

假定你已经通过 WebGUI 创建资源池“dev-pool”，现在我们可以向该资源池赋予访问权限：

pveum aclmod /pool/dev-pool/ -group developers -role PVEAdmin

现在我们的软件开发部门就可以管理该资源池中的资源了。