1. Các khái niệm liên quan tới keystone
2. Keystone là gì

Keystone là dịch vụ openstack cung cấp API xác thực ứng dụng khách, phát hiện dịch vụ,phân quyền cho nhiều người thuê bằng cách triển khai Openstack identity API

1. Project

Trong Keystone, Project được dùng bởi các services của OpenStack để nhóm và cô lập các nguồn tài nguyên. Nó có thể được hiểu là 1 nhóm các tài nguyên mà chỉ có một số các user mới có thể truy cập và hoàn toàn tách biệt với các nhóm khác.

Mục đích cơ bản nhất của keystone chính là nơi để đăng kí cho các projects và xác định ai được phép truy cập projects.

Bản thân projects không sở hữu users hay groups mà users và groups được cấp quyền truy cập tới project sử dụng cơ chế gán role.

Việc gán role cho user còn được gọi là "grant".

1. Domain

Domain được định nghĩa là một tập hợp các users, groups, và projects. Nó cho phép người dùng chia nguồn tài nguyên cho từng tổ chức sử dụng mà không phải lo xung đột hay nhầm lẫn.

1. Actor

Users và User Groups là những đối tượng được cấp phép truy cập tới các nguồn tài nguyên được cô lập trong domains và projects.

Groups là một tập hợp các users. Users và User Groups được gọi là Actors.

Mối quan hệ giữa domains, projects, users, và groups: users và groups được cấp quyền truy cập tới project sử dụng cơ chế gán role.

1. Role

Roles được dùng để hiện thực hóa việc cấp phép trong keystone. Một actor có thể có nhiều roles đối với từng project khác nhau.

1. Target

Projects và Domains đều giống nhau ở chỗ cả hai đều là nơi mà role được "gán" lên.

Vì thế chúng được gọi là targets.

1. Token

Phần sau sẽ nói chi tiết hơn

1. Catalog

Chứa URLs và endpoints của các services khác nhau.

Nếu không có Catalog, users và các ứng dụng sẽ không thể biết được nơi cần chuyển yêu cầu để tạo máy ảo hoặc lưu dữ liệu.

service này được chia nhỏ thành danh sách các endpoints và mỗi một endpoint sẽ chứa admin URL, internal URL, and public URL.

1. Credentials

Là thông tin dùng để xác thực người dùng. Ví dụ như username, password và API key, hay là token mà được cung cấp.

1. Assignment

Role assignment là sự kết hợp của actor, target và role.

Role assignment được cấp phát, thu hồi, và có thể được kế thừa giữa các users, groups, project và domains.

1. Endpoint

Là một địa chỉ truy cập mạng, thường là địa chỉ URL của một dịch vụ để từ đó truy cập vào dịch vụ.

1. Identity service
2. User management
3. Token
4. Token là gì
5. Các loại token
6. UUID token

UUID là token format đầu tiên của Keystone, nó đơn giản chỉ là một chuỗi UUID gồm 32 kí tự được generate ngẫu nhiên. Nó được xác thực bởi Identity service.

UUID token buộc phải được lưu lại trong một backend (thường là database). Nó cũng có thể được loại bỏ bằng cách sử dụng DELETE request tới token id. Tuy nhiên nó sẽ không thực sự bị loại bỏ khỏi backend mà chỉ được đánh dấu là đã được loại bỏ. Vì nó chỉ có 32 bytes nên kích thước của nó trong HTTP header cũng sẽ là 32 bytes.

Loại token này rất nhỏ và dễ sử dụng, tuy nhiên nếu sử dụng nó, Keystone sẽ là "cổ chai" của hệ thống bởi mọi cuộc giao tiếp đều cần tới Keystone để xác thực token.

**Sinh mã:**

User request tới Keystone tạo token với các thông tin: username, password, project name.

Chứng thực user, lấy User ID từ backend LDAP (dịch vụ Identity)

Chứng thực project, thu thập thông tin Project ID và Domain ID từ backend SQL (dịch vụ Resources)

Lấy ra Roles từ backend trên project hoặc domain tương ứng trả về cho user, nếu user không có bất kì role nào thì trả về Failure (dịch vụ Assignment)

Thu thập các Services và các Endpoint của các service đó (Dịch vụ Catalog)

Tổng hợp các thông tin về Identity, Resources, Assignment, Catalog ở trên đưa vào Token payload, tạo ra token sử dụng hàm uuid.uuid4().hex.

Lưu thông tin của token vào SQL/KVS backend với các thông tin: TokenID, Expiration, Valid, UserID, Extra

**Xác thực mã:**

Gửi yêu cầu chứng thực token sử dụng API: GET v3/auth/tokens và token (X-Subject-Token, X-Auth-Token)

Thu thập token payload từ token backend KVS/SQL kiểm tra trường valid. Nếu không hợp lệ trả về thông báo Token Not Found. Nếu tìm thấy chuyển sang bước theo.

Phân tích token và thu thập metadata: User ID, Project ID, Audit ID, Token Expire

Kiểm tra token đã hết hạn chưa. Nếu thời điểm hiện tại < expired time theo UTC thì token chưa hết hạn, chuyển sang bước tiếp theo, ngược lại trả về thông báo token not found.

Kiểm tra xem token đã bị thu hồi chưa (kiểm tra trong bảng revocation\_event của database Keystone). Nếu token đã bị thu hồi (tương ứng với 1 event trong bảng revocation\_event) trả về thông báo Token Not Found. Nếu chưa bị thu hồi trả về token (truy vấn HTTP thành công HTTP/1.1 200 OK)

**Thu hồi mã:**

Gửi yêu cầu thu hồi token với API request DELETE v3/auth/tokens. Trước khi thực hiện sự kiện thu hồi token thì phải chứng thực token nhờ vào tiến trình Validation Token.

Kiểm tra trường Audit ID. Nếu có, tạo sự kiện thu hồi với audit ID. Nếu không, tạo sự kiện thu hồi với token expired

Nếu tạo sự kiện thu hồi token với audit ID, các thông tin cần cập nhật vào revocation\_event table của Keystone database gồm: audit\_id, revoke\_at, issued\_before.

Nếu tạo sự kiện thu hồi token với audit ID, các thông tin cần cập nhật vào revocation\_event table của Keystone database gồm: user\_id, project\_id, issued\_before, token\_expired

Loại bỏ các sự kiện của các token đã hết hạn từ bảng revocation\_event của database Keystone.

Cập nhật vào token database, thiết lập lại trường "valid" thành false (0).

**Mô hình hoạt động**

Client cung cấp user/password.

Keystone tạo một token UUID. Lưu trữ các thẻ UUID ở backend. Gửi một bản sao của UUID token cho khách hàng.

Các khách hàng sẽ cache token.

UUID sau đó sẽ được thông qua cùng với mỗi cuộc gọi API của khách hàng.

Mỗi khi có yêu cầu của người dùng, các thiết bị đầu cuối API sẽ gửi UUID này trở lại Keystone để xác nhận.

Keystone sẽ trả về "thành công" hoặc thông báo "thất bại" đến điểm cuối API.

1. PKI token

Token này chứa một lượng khá lớn thông tin ví dụ như: thời gian nó được tạo, thời gian nó hết hiệu lực, thông tin nhận diện người dùng, project, domain, thông tin về role cho user, danh mục dịch vụ,... Tất cả các thông tin này được lưu ở trong Cryptographic Message Syntax (CMS). Với PKIz thì phần payload được nén sử dụng zlib.

**Sinh mã:**

Người dùng gửi yêu cầu tạo token với các thông tin: Username, Password, Project Name

Keystone sẽ chứng thực các thông tin về Identity, Resources và Assignment (định danh, tài nguyên, assignment)

Tạo token payload định dạng JSON

Sign JSON payload với Signing Key và Signing Certificate, sau đó được đóng gói lại dưới dạng CMS.

Bước tiếp theo, nếu muốn đóng gói token định dạng PKI thì convert payload sang UTF-8, convert token sang một URL định dạng an toàn. Nếu muốn token đóng gói dưới định dạng PKIz, thì phải nén token sử dụng zlib, tiến hành mã hoá base64 token tạo ra URL an toàn, convert sang UTF-8 và chèn thêm tiếp đầu ngữ "PKIz"

Lưu thông tin token vào backend

**Xác thực mã:**

Vì id được generate bằng hàm hash của token nên quá trình validate token sẽ bắt đầu bằng việc sử dụng hàm hash để băm PKI token. Các bước sau đó (validate trong backend,...) hoàn toàn giống với uuid.

**Thu hồi mã:**

Hoàn toàn tương tự như tiến trình thu hồi UUID token

**Mô hình hoạt động**

Với thẻ PKI/PKIz, Keystone trở thành một Certificate Authority (CA). Nó sử dụng signing key và certificate (không mã hóa) để ký token của user.

Mỗi điểm cuối API giữ một bản sao của Keystone của:

Signing certificate.

Revocation list.

CA certificate.

Các thiết bị đầu cuối API sử dụng các bit để xác nhận các yêu cầu sử dụng. Không cần cho yêu cầu trực tiếp đến Keystone với từng xác nhận. Những gì được xác nhận là chữ ký Keystone đặt trên thẻ người dùng và danh sách thu hồi Keystone của. Điểm cuối API sử dụng các dữ liệu trên để thực hiện quá trình này offline.

1. Fernet token

Fernet token được sử dụng đầu tiên tại Openstack Kilo. Không giống như các loại token trước, Fernet không yêu cầu sử dụng backend

Fernet token chứa một lượng nhỏ dữ liệu ví dụ như thông tin để nhận diện người dùng, project, thời gian hết hiệu lực,... Nó được ký số bởi symmetric key để ngăn ngừa việc giả mạo.

Dữ liệu trong Fernet token được bảo vệ bởi các thuật toán mã hoá

Thư mục chứa Fernet key: /etc/keystone/fernet-keys/

Fernet key được dùng trong việc mã hoá và giải mã các Fernet token

Thực hiên tất cả mã hoá bằng cách sử dụng khoá đầu tiên trong danh sách và cố gắng giải mã bằng tất cả các khoá từ danh sách đó.

Trạng thái của key xác định Keystone sẽ sử dụng key đó làm gì với fernet token. Các loại khác nhau như sau:

Primary key: Dùng để mã hoá và giải mã token. Key này luôn luôn có chỉ số lớn nhất trong repo.

Secondary key: Khi có một primary key mới, primary key cũ sẽ trở thành secondary key. Sử dụng để giải mã token. Các secondary key có chỉ số nằm giữa 0 và primary key.

Staged key: Staged key là một loại key đặc biệt có một số điểm giống với secondary key. Chỉ có duy nhất một staged key trong repo. Giống như secondary key, staged key dùng để giải mã token. Staged key sẽ trở thành primary key trong lần rotate tiếp theo. Staged key luôn có file name là 0 trong key repo.

**Fernet key rotate**

Bước đầu chưa có key repo sẽ khởi tạo một key repo bằng keystone-manager để tạo 2 khoá trong repo. Khi 2 file được tạo thì sẽ có file tên 1 là primary key và file 0 sẽ là staged key và không có secondary key.

Lần rotate đầu tiên, staged 0 sẽ trở thành primary key 2, primary key 1 trở thành secondary key và staged mới sẽ được tạo ra với file name là 0.

Lần rotate thứ 2, staged 0 trở thành primary key 3, primary key 2 trở thành secondary key 2, secondary 1 giữ nguyên và tạo ra staged key mới.

Khi sử dụng fernet token yêu cầu chú ý về thời hạn của token và vòng đời của khoá. Vấn đề nảy sinh khi secondary key bị xoá khỏi key repo trong khi vẫn cần dùng key đó để giải mã một token chưa hết hạn (token này được mã hoá bởi key đã bị xoá). Để giải quyết vấn đề này, trước hết cần lên kế hoạch xoay khoá.

Ví dụ bạn muốn token hợp lệ trong vòng 24 giờ và muốn xoay khoá cứ mỗi 6 giờ. Như vậy để giữ 1 key tồn tại trong 24 giờ cho mục đích decrypt thì cần thiết lập option max\_active\_keys=6 trong file cấu hình keystone. Điều này giúp cho việc giữ tất cả các key cần thiết nhằm mục đích xác thực token mà vẫn giới hạn được số lượng key trong key repo.

**Sinh mã fernet**

Các thông tin được cho vào Token Payload và được Padding cho đủ khối kích thước.

Dùng Encrypting key để mã hóa khối trên và chuyển vào Ciphertext.

Các trường Version, Timestamp, IV do hệ thống tự tạo ra.

Dùng Signing key(SHA) để mã hóa 4 trường Version, Timestamp, IV, Ciphertext sau đó chuyển vào HMAC.

Fernet Format Version (0x80): 8 bits, biểu thị phiên bản của định dạng token

Current Timestamp: Số nguyên 64-bits không dấu, chỉ nhãn thời gian tính theo giây, tính từ 01/01/1970, chỉ ra thời điểm token được tạo ra.

Initialization Vector (IV): Key 128-bits sử dụng mã hoá AES và giải mã Ciphertext.

Ciphertext: Là keystone payload kích thước biến đổi tuỳ vào phạm vi của token. Cụ thể hơn, với token có phạm vi project, Keystone payload bao gồm: version, user id, method, project id, expiration time, audit id.

HMAC: 256-bits SHA256 HMAC (Keyed-Hash Message Authentication Code) - Mã xác thực thông báo sử dụng hàm một chiều có khoá với signing key kết nối 4 trường ở trên.

**Xác thực mã fernet**

Gửi yêu cầu xác thực token với API GET v3/auth/tokens

Khôi phục lại padding, trả lại token với padding chính xác

Decrypt sử dụng Fernet Keys để thu lại token payload

Xác định phiên bản của token payload (Unscoped token: 0, Domain-scoped: 1, Project-scoped: 2)

Tách các trường của payload để chứng thực. Ví dụ với token phạm vi project gồm các trường sau: user id, project id, method, expiration, audit id.

Kiểm tra xem token đã hết hạn chưa. Nếu thời điểm hiện tại lớn hơn so với thời điểm hết hạn thì trả về thông báo "Token not found". Nếu token chưa hết hạn thì chuyển sang bước tiếp theo.

Kiểm tra xem token đã bị thu hồi chưa. Nếu token đã bị thu hồi (tương ứng với 1 sự kiện thu hồi trong bảng revocation\_event của database keystone) thì trả về thông báo "Token not found". Nếu chưa bị thu hồi thì trả lại token (thông điệp phản hồi thành công HTTP/1.1 200 OK)

1. Keystone workflow

có được mã thông báo chưa được đăng ký với bài đăng thông tin xác thực.

khám phá các dự án bạn có quyền truy cập.

có được mã thông báo phạm vi dự án bằng thông tin đăng nhập của bạn hoặc mã thông báo chưa đăng ký từ bước 1.

gọi ra dịch vụ đích bằng cách gửi yêu cầu đến các điểm cuối trong mã thông báo từ bước 3.

xác thực vai trò và truy cập siêu dữ liệu trong mã thông báo với dịch vụ keystone hoặc phần mềm trung gian keystone.

phục vụ yêu cầu API.

trả lại phản hồi.

1. Nova
2. Nova là gì

Nova là dự án OpenStack cung cấp các phiên bản máy tính (hay còn gọi là máy chủ ảo). Nova hỗ trợ tạo máy ảo, máy chủ baremetal (thông qua việc sử dụng ironic) và hỗ trợ hạn chế cho các vùng chứa hệ thống. Nova chạy như một tập hợp các daemon trên các máy chủ Linux hiện có để cung cấp dịch vụ đó.

1. **Cấu trúc của nova**

- nova-api (service): Tiếp nhận và phản hồi API call, HTTP request. Chuyển đổi các lệnh và giao tiếp với các thành phần khác thông qua oslo.messaging hoặc HTTP.

- nova-api-metadata (service): Tiếp nhận yêu cầu lấy metadata từ instance.

- nova-compute (service): Quản lý instance thông qua các Hypervisor API. Về cơ bản nó tiếp nhận các hành động từ hàng đợi và thực hiện một chuỗi các lệnh vận hành VM và cập nhật trạng thái của VM vào database.

- nova-scheduler (service): Nó lấy các yêu cầu tạo VM từ hàng đợi và xác định xem VM sẽ được tạo ở Host nào.

- nova-conductor (module): Là module trung gian giữa nova-compute và CSDL. Nó hủy tất cả các yêu cầu trực tiếp vào CSDL của nova-compute.

- nova-network : tiếp nhận yêu cầu về network từ hàng đợi và control network.

- nova-novncproxy (daemon): Cung cấp proxy để truy cập VM thông qua VNC. Hỗ trợ novnc client trên trình duyệt.

- nova-spicehtml5proxy (daemon): Cung cấp proxy để truy cập VM thông qua SPICE. Hỗ trợ client chạy trên trình duyệt HTML5.

- nova-xvpvncproxy (daemon): Cung cấp proxy để truy cập VM thông qua VNC.

- nova-consoleauth (daemon): Ủy quyền token cho người dùng mà console proxy cung cấp.

- The queue: Trung tâm chuyển tiếp bản tin giữa các daemon. Cung cấp bởi 1 phần mềm message queue hỗ trợ giao thức AMQP như RabbitMQ, ZeroMQ.

1. **Quá trình tạo instance**

- Bước 1: Từ Dashboard hoặc CLI, nhập thông tin chứng thực (ví dụ: user name và password) và thực hiện lời gọi REST tới Keystone để xác thực

- Bước 2: Keystone xác thực thông tin người dùng và tạo ra một token xác thực gửi trở lại cho người dùng, mục đích là để xác thực trong các bản tin request tới các dịch vụ khác thông qua REST

- Bước 3: Dashboard hoặc CLI sẽ chuyển yêu cầu tạo máy ảo mới thông qua thao tác "launch instance" trên openstack dashboard hoặc "nova-boot" trên CLI, các thao tác này thực hiện REST API request và gửi yêu cầu tới nova-api

- Bước 4: nova-api nhận yêu cầu và hỏi lại keystone xem auth-token mang theo yêu cầu tạo máy ảo của người dùng có hợp lệ không và nếu có thì hỏi quyền hạn truy cập của người dùng đó.

- Bước 5: Keystone xác nhận token và update lại trong header xác thực với roles và quyền hạn truy cập dịch vụ lại cho nova-api

- Bước 6: nova-api tương tác với nova-database

- Bước 7: Dababase tạo ra entry lưu thông tin máy ảo mới

- Bước 8: nova-api gửi rpc.call request tới nova-scheduler để cập nhật entry của máy ảo mới với giá trị host ID (ID của máy compute mà máy ảo sẽ được triển khai trên đó). (Chú ý: yêu cầu này lưu trong hàng đợi của Message Broker - RabbitMQ)

- Bước 9: nova-scheduler lấy yêu cầu từ hàng đợi

- Bước 10: nova-scheduler tương tác với nova-database để tìm host compute phù hợp thông qua việc sàng lọc theo cấu hình và yêu cầu cấu hình của máy ảo

- Bước 11: nova-database cập nhật lại entry của máy ảo mới với host ID phù hợp sau khi lọc.

- Bước 12: nova-scheduler gửi rpc.cast request tới nova-compute, mang theo yêu cầu tạo máy ảo mới với host phù hợp.

- Bước 13: nova-compute lấy yêu cầu từ hàng đợi.

- Bước 14: nova-compute gửi rpc.call request tới nova-conductor để lấy thông tin như host ID và flavor(thông tin về RAM, CPU, disk) (chú ý, nova-compute lấy các thông tin này từ database thông qua nova-conductor vì lý do bảo mật, tránh trường hợp nova-compute mang theo yêu cầu bất hợp lệ tới instance entry trong database)

- Bước 15: nova-conductor lấy yêu cầu từ hàng đợi

- Bước 16: nova-conductor tương tác với nova-database

- Bước 17: nova-database trả lại thông tin của máy ảo mới cho nova-conductor, nova condutor gửi thông tin máy ảo vào hàng đợi.

- Bước 18: nova-compute lấy thông tin máy ảo từ hàng đợi

- Bước 19: nova-compute thực hiện lời gọi REST bằng việc gửi token xác thực tới glance-api để lấy Image URI với Image ID và upload image từ image storage.

- Bước 20: glance-api xác thực auth-token với keystone

- Bước 21: nova-compute lấy metadata của image(image type, size, etc.)

- Bước 22: nova-compute thực hiện REST-call mang theo auth-token tới Network API để xin cấp phát IP và cấu hình mạng cho máy ảo

- Bước 23: quantum-server (neutron server) xác thực auth-token với keystone

- Bước 24: nova-compute lấy thông tin về network

- Bước 25: nova-compute thực hiện Rest call mang theo auth-token tới Volume API để yêu cầu volumes gắn vào máy ảo

- Bước 26: cinder-api xác thực auth-token với keystone

- Bước 27: nova-compute lấy thông tin block storage cấp cho máy ảo

- Bước 28: nova-compute tạo ra dữ liệu cho hypervisor driver và thực thi yêu cầu tạo máy ảo trên Hypervisor (thông qua libvirt hoặc api - các thư viện tương tác với hypervisor)