Lecture 6

OpenStack Keystone

OpenStack Internals

The primary sub-parts of the OpenStack Cloud software:

	Service	Comments
nova	Compute	Manages pools of computer resources
glance	Image	Disk and server images
keystone	Identity	Common authentication system
horizon	Dashboard	GUI
cinder	Block Storage	Persistent block storage devices
swift	Object Storage	Object and file management
neutron	Networking	IP addresses, VLANS, etc.
trove	Database	Provisions relational and non-relational DBs
heat	Orchestration	Launches composite cloud applications based on templates, specify relationships between resources
ironic	Bare metal provisioning	Provisions and turns on and off bare metal machines instead of virtual machines
ceilometer	telemetry	Billing system
sahara	Elastic Map Reduce	Handles Hadoop clusters
zaqar	Multiple Tenant Cloud Messaging	Can be used in SaaS to send messages between multiple components of the SaaS

OpenStack Cloud: Keystone

Keystone is a common authentication system used by all the OpenStack services. It provides: Identity, Token, Catalog, and Policy services

Allows different kinds of authentication, these include:

1. Username/password
2. Token based
3. Amazon Web Services

Keystone is accessed using the OpenStack Identity API

OpenStack Cloud (cont’d): Keystone (cont’d)

Keystone services include:

1. Identity
   - Credential validation (Username/password)
   - Maintains users, groups, projects, domains, roles
2. Token
   - After credentials have been verified, tokens for authentication are generated/validated/managed.
3. Catalog
   - An endpoint registry used for endpoint discovery
4. Policy
   - Rule-based authorization, rule management

OpenStack Cloud (cont’d): Keystone (cont’d)

Data types recognized by Keystone:

- User—has account credentials, has project or domain
- Group—collection of users, has project or domain
- Project—unit of ownership, has one or more Users
- Domain—unit of ownership, has Users, Groups, Projects
- Role—a descriptive metadata item, with User/Project pair
- Token—identifying credential, used with User or User/Project
- Extras—key/value metadata, used with User/Project pair
- Rule—a set of requirements for performing an action

OpenStack Cloud (cont’d): Keystone (cont’d)

Keystone must be configured before use.

Keystone allows plug-ins to extend authentication mechanisms.

Basic Keystone: Authentication

Keystone provides:

- token
- Service catalog with information about the service endpoints for which the user is authorized

Basic Keystone: Authentication (cont’d)

- Each key pair has two parts:
  - Public key
  - Private key
- The Public key is registered in OpenStack
- The Private key is stored in a .pem file on local machine

Keystone Token Formats

- Keystone allows different kinds of authentication, these include:
  - Username/password
  - Token based
  - Amazon Web Services

Keystone Token Formats

- Keystone can be configured to use one of these different token formats:
  - Universally Unique Identifier (UUID)—version 4
  - PKI
  - PKIZ
  - Fernet

Keystone Token Formats (cont’d)

- A token is valid only for a limited time
  - So if it is stolen it is only a short time the thief is able to use it

Keystone Token Formats (cont’d)

- Token size is a major issue:
  - Performance degrades when huge amounts of token data is stored in persistent token database
    - Authentication requests require searching for the token in persistent storage, and can take a very long time
  - Results in the need to repeatedly flush the persistent token database
  - Some token formats are larger than others, which makes the problem worse

Keystone Token Formats (cont'd)

- Why would one wish to revoke a token?
- Fisher (2015) gives the following reasons:
  - A user has been deleted
  - A role has been removed from a user
  - A user has been removed from a project
  - A user has logged out of the OpenStack Dashboard (Horizon)
  - A user has switched to a different project in the OpenStack Dashboard (Horizon)

Keystone Token Formats (cont’d)

- If a user has been removed for bad behavior or leaving the company, it is important to be able to remove the token immediately
- Default token expiration time is 3600 seconds (one hour)

Keystone Token Formats (cont’d)

- In Keystone, a revocation event is stored on a revocation event list
- Every time a token is validated, this list is checked
- Every time any token is checked, keystone removes expired tokens from the revocation event list

Universally Unique Identifier (UUID)

- A UUID is a 128 bit number (16 bytes/octets)
  - An example would be:
    - abcd1234-fae3-12cd-a4be-1234abcd4321
  - 8 digits, 4 digits, 4 digits, 4 digits,12 digits
  - Lower case hexadecimal
  - The digit in red in the 3rd group shows the version # (1, 2, 3, 4, or 5)
  - The first two bits of the digit in red in the 4th group show the variant. “10” is the variant from RFC4122, so this digit would be 8, 9, a, or b

Universally Unique Identifier (UUID)

- Variants from RFC 4122:

MSB	MSB -1	MSB -2	Description
0	X	X	Network Computing System (NCS) backward compatibility
1	0	X	RFC 4122 Variant
1	1	0	Microsoft backward compatibility
1	1	1	Reserved for the future

Universally Unique Identifier (UUID)-cont’d

- There are 5 versions of UUIDs:
  1. Time-based+MAC address
  2. DCE
  3. Name-based with MD5 hash
  4. Random (what Keystone uses)
  5. Name-based with SHA-1 hash

MD5 Message Digest Hash and SHA-1 Hash

- Even small changes in the message will (usually) result in a mostly different hash
- MD-5 Hash
  - 32 digit hexadecimal number
- SHA-1 Hash
  - 40 digit hexadecimal number
- SHA-1 considered safer than MD-5

Version 1: Time-Based UUID

- Timestamp is 60 bit unsigned integer (15 hex digits), Coordinated Universal Time, representing 100 nanosecond intervals since 15 Oct 1582 (date of Gregorian calendar reform).
- Clock ID is a 14 bit unsigned integer, initially (once in a system lifetime) initialized to a random number.
- MAC address is 48 bit unsigned integer, in 6 sets of two hexadecimal digits

Version 1: Time-Based UUID (cont'd)

- Given: abcd1234-fae3-12cd-a4be-1234abcd4321
- Time is: 0x2cdfae3abcd1234
  - Time_hi = 0x2cd, Time mi=0xfae3, Time_low=0xabcd1234
- Clock ID is: 10 0100 1011 1110 base 2
  - Clock hi is 10 0100 (MSB is version 10, gives a4) Clock lo is 0xbe
- MAC address is: 12:34:ab:cd:43:21

Byte #	0-3	4-5	6-7	8	9	10-15
	Time_low	Time_ mid	Time_hi & version	Clock hi & reserv ed	Clock low	Node (MAC) address
	abcd1234	fae3	12cd	a4	be	1234abcd4321

Version 4: Random UUID

xxxxxxxx-xxxx-**4**xxx-**a**xxx-xxxxxxxxxxxx

- Where the red 4 indicates the version number (as before)
- The red a indicates the variant (as before, the first two bits are “10” which means this value can be 8, 9, a, or b)
- The other hex digits, represented by x, are randomly generated.

This is the version of UUID that Keystone uses

PKI/PKIZ -- some terminology

Cryptographic Message Syntax (CMS) from RFC 5652:

- used to digitally sign, digest, authenticate, or encrypt message content

Certificate Signing Request (CSR)

- A message sent from an applicant to a certificate authority in order to apply for a digital identity certificate

Private Key (yourkey.pem), CA Certificate (ca.pem), Signing Key (signing_key.pem), Signing Certificate (signing_cert.pem)

PKI/PKIZ (cont'd)

Payload Format (originally in JSON format, see below)	
when issued	User ID
when expires	Role Id for this user
Project ID	Domain ID
Expiration time	Service catalog
Signed using Cryptographic Message Syntax	
Base 64url encoded	

Prior to encryption, payload is represented in JSON format:

```
{ "access" : { "metadata":…},
  "serviceCatalog":[…endpoints…]},
  "token": {…},
  "user: {…}
}
```

- Note entire service catalog is included

PKI/PKIZ (cont’d)

- A basic token with a single endpoint is approx. 1700 bytes
- Token sizes increase proportionally as regions and services are added to the catalog.
    - Sometimes over 8K bytes

PKIZ is PKI compressed using zlib compression

PKI/PKIZ (cont'd)

Keystone acts as a Certificate Authority

PKIZ is PKI compressed using zlib compression

PKI/PKIZ (cont’d)

Briefly, when Entity A wishes to connect to Entity B on a network:

- a third party that is trusted by both entities verifies Entity A’s credentials
- so Entity B will know that Entity A is who it says it is
- the trusted third party is called a “Certification Authority”

PKI/PKIZ (cont'd)

- After checking out Entity A's credentials, the Certification Authority issues a certificate
  - that it digitally signs
  - that says that Entity A is who it says it is
- Entity A can then pass this certificate on to Entity B
- Entity B recognizes the Certification Authority's digital signature on the certificate
  - so it accepts the certificate and is willing to talk to Entity A

PKI/PKIZ (cont'd)

- When using PKI tokens, Keystone acts as a Certification Authority.
- At the time Keystone is installed,the following are generated:
  - Certificate Authority private key
  - Certificate Authority certificate
  - Signing Private Key
  - Signing Certificate

PKI/PKIZ (cont’d)

- When you’re using either PKI or PKIZ tokens the entire normal Keystone validation response is included as part of the token. This includes (among other items):
  - Service catalog
  - User’s roles
  - User’s public key
  - Token expiration date

PKI/PKIZ (cont’d)

- This information is now converted into Cryptographic Message Syntax (CMS) and signed with the Certificate Authority Signing Key
- The recipient will of course have to convert it back from CMS and verify the signature before processing it further

PKI/PKIZ (cont’d)

- When a user sends a request including a PKI token to an OpenStack component service via its API, the OpenStack component service/API has a local copy of:
  - Signing Certificate
  - Certificate Authority Certificate
  - Certification Revocation List (CRL)
- If the user’s PKI token is valid, and the token has not yet expired
  - then the OpenStack API will process the user’s request

PKI/PKIZ (cont’d)

- PKI tokens can be extremely large:
  - A basic token with a single endpoint is approx. 1700 bytes
- The PKI token sizes increase proportionally as regions and services are added to the catalog:
  - sometimes can be over 8K bytes
- PKIZ tokens try to fix this problem by compression—PKIZ tokens are just PKI tokens compressed using zlib compression

PKI/PKIZ (cont'd)

- PKI tokens can be extremely large:
  - A basic token with a single endpoint is approx. 1700 bytes
- The PKI token sizes increase proportionally as regions and services are added to the catalog:
  - sometimes can be over 8K bytes
- PKIZ tokens try to fix this problem by compression—PKIZ tokens are just PKI tokens compressed using zlib compression

PKI/PKIZ (cont’d)

- Since the OpenStack component service (endpoint/API) has local copies of the certificates, it doesn’t have to go back to Keystone to ask if the token is valid every time it receives a user request.
  - It only has to download the Signing Certificate and the Certificate Authority Certificate once, at the time they are first needed.
  - However, periodically each OpenStack component service must update the Certificate Revocation List (CRL) by requesting it from Keystone.
  - This must be done frequently
    - once per second is the number that Kupidura (2013) mentions

PKI/PKIZ (cont’d)

- PKIZ tokens, even though compressed, are still quite long
- Young (2014) says:

  “The packaging for a PKIZ token has a lower bound based on the signing algorithm. An empty CMS document of compressed data is going to be no less than 650 bytes. An unscoped token with proper compression comes in at 930 bytes.”

PKI/PKIZ (cont'd)

- So based on certificate checking being done locally to the OpenStack component service (endpoint), the PKI and PKIZ token implementations result in reduced network traffic compared to UUID tokens.

- However, since the PKI and PKIZ tokens themselves are very long, network traffic due just to sending these long tokens can still be a problem

Fernet Tokens

- Fernet tokens use Symmetric Key Encryption (shared private keys)
  - using fernet tokens avoids having to store or replicate tokens in a database
  - available Kilo release (April 2015) of OpenStack and later

Fernet Tokens (cont'd)

- Fernet tokens employs key rotation:
  - Uses a list of symmetric keys:
    - stored in a key repository:
      - /etc/keystone/fernet-keys
      - fernet key repository is file-based (each key is a file)
    - the max_active_keys configuration option sets list size (default is 3)
    - should be accessible by keystone only

Fernet Tokens (cont'd)

Fernet Key File Contents	
Signing Key	Encrypting Key
first 16 bytes	Last 16 bytes
key that will be used to do HMAC SHA256 signing of token contents	key that will be used to encrypt token payload using AES encryption
128 bits 32 bytes (total)	
Base 64url encoded	

Fernet Tokens (cont'd)

Fernet Token Contents		
0x80	Fernet Token Version	8 bits
Current Timestamp	In UTC	64 bits
Initialization Vector	Random Number	128 bits
Payload/Ciphertext		• variable size, • a multiple of 128 bits • padded as necessary
HMAC	Signed using Signing Key—SHA256	
Base 64url encoded		

Fernet Tokens (cont'd)

Payload/Ciphertext Format	
Version	• unscoped=0 • domain scoped=1 • project scoped=2 • trust scoped=3 • federated unscoped=4 • federated domain scoped =6 • federated project scoped=5
User Identifier	UUID
Methods	oauth1, password, token, external
Project identifier	
Expiration time	In UTC
Audit Identifier	• serves as a Token Identifier • url safe random number
padding, as needed	
Encrypted using AES with Encrypting Key	
Base 64url encoded	

Fernet Tokens (cont’d)

- Primary Key:
  - the key that does all encryption
  - Can also do decryption.
  - the key with the highest index
- Staged Key:
  - will be the next signing key, after rotation occurs
  - each new key starts out as the Staged Key
  - the key with index 0
- Secondary Key:
  - used to be the Primary Key
  - now can only do decryption
  - all keys in between 0 and the highest key

Fernet Tokens (cont’d)--Rotation

Example of Rotation, max_active_keys=3 (Fischer, 2015)(Blagstad 2015):

State	Staged Key	Secondary Key	Primary Key (Signing Key)
Initial	Key0	Key1	Key2
Post Rotation	new Key0	Key2 (Key1 was deleted)	Key3 (was old Key0)

Fernet Tokens (cont’d)

- At initial state, do:

  ```
  ls /etc/keystone/fernet-keys
  ```

- get

  ```
  0 1 2
  ```

- After rotate, do:

  ```
  ls /etc/keystone/fernet-keys
  ```

- get

  ```
  0 2 3
  ```

Fernet Tokens (cont'd)--Rotation

Example of Rotation, max_active_keys=4 (Desai and Pokorny 2015:

State	Staged Key	Secondary Key	Secondary Key	Primary Key (Signing Key)
Initial	Key0	No secondary key		Key1
Post Rotation	new Key0		Key1	Key2 (was old Key0)
Post Rotation	new Key0	Key1	Key2	Key3 (was old Key0)

Fernet Tokens (cont'd)

Example Fernet Token (Mathews (2016):

```
gAAAAABU7roWGiCuOvgFcckec0ytpGnMZDBLG9hA7
Hr9qfvdZDHjsak39YN98HXxoYLIqVm19Egku5YR3wyI7
heVrOmPNEtmrfIM1rtahudEdEAPM4HCiMrBmiA1Lw6S
U8jc2rPLC7FK7nBCia_BGhG17NVHuQu0S7waA306jyK
NhHwUnpsBQ%3D
```

Fernet Tokens (cont’d)

- Fernet tokens have lifespans:
  - If you plan to rotate your primary key every 30 minutes and each fernet token has a lifespan of 6 hours, then max_active_keys must be >=12 (Bradstad, 2015)
- To setup a new key repository,do:
  - keystone-manage fernet_setup
- To perform a rotate, do:
  - keystone-manage fernet_rotate
    - This will create a new key0 to take part in the rotation
- Using key rotation allows the operator to distribute the new key over a span of time

Fernet Tokens (cont’d)

- Advantages compared to UUID tokens
  - Pruning of tokens from the token database is no longer required
    - It is done automatically as part of rotation
  - As long as all keystone nodes share the same key repository, fernet tokens across keystone nodes can be created and validated immediately
  - Some benchmarks have shown Fernet tokens up to 85% faster than UUID tokens (Mathews, 2016)

Fernet Tokens (cont’d)

- Advantages compared to PKI/PKIZ tokens:
  - Pruning of tokens from the token database is no longer required
    - It is done automatically as part of rotation
  - As long as all keystone nodes share the same key repository, fernet tokens across keystone nodes can be created and validated immediately
  - Fernet tokens are much smaller than PKI and PKIZ tokens
    - PKI and PKIZ tokens usually > 1600 bytes
    - Fernet tokens <= 256 bytes
  - Some benchmarks have shown Fernet tokens up to 89% faster than PKI/PKIZtokens (Mathews, 2016)

Fernet Tokens (cont’d)

- Hash-based Message Authentication Code (HMAC)
  - HMAC (key, message) =
    - hash (

```
(key XOR outer-padding) concatenated with
          hash(
                    (key XOR inner padding) concatenated with
                     message
          )
)
```

  - where
    - outer padding = 0x5c0x5c…0x5c (block long hex constant)
    - Inner padding = 0x360x36…0x36 (block long hex constant)

Fernet Tokens (cont'd)

- For HMAC on Fermat tokens:
  - Key used is Signing Key
  - Hash function used is SHA-256
    - The block size the algorithm works on is 512 bits
    - Output size is 256 bits (32 bytes)

Fernet Tokens (cont'd)--Format

- A base 64 encoding of the signing key and the encrypting key.

Signing Key **SHA256 HMAC** **128 bits** **(16 bytes)**	**Encrypting Key** **AES** **128 bits** **(16 bytes)**

Fernet Tokens (cont’d)

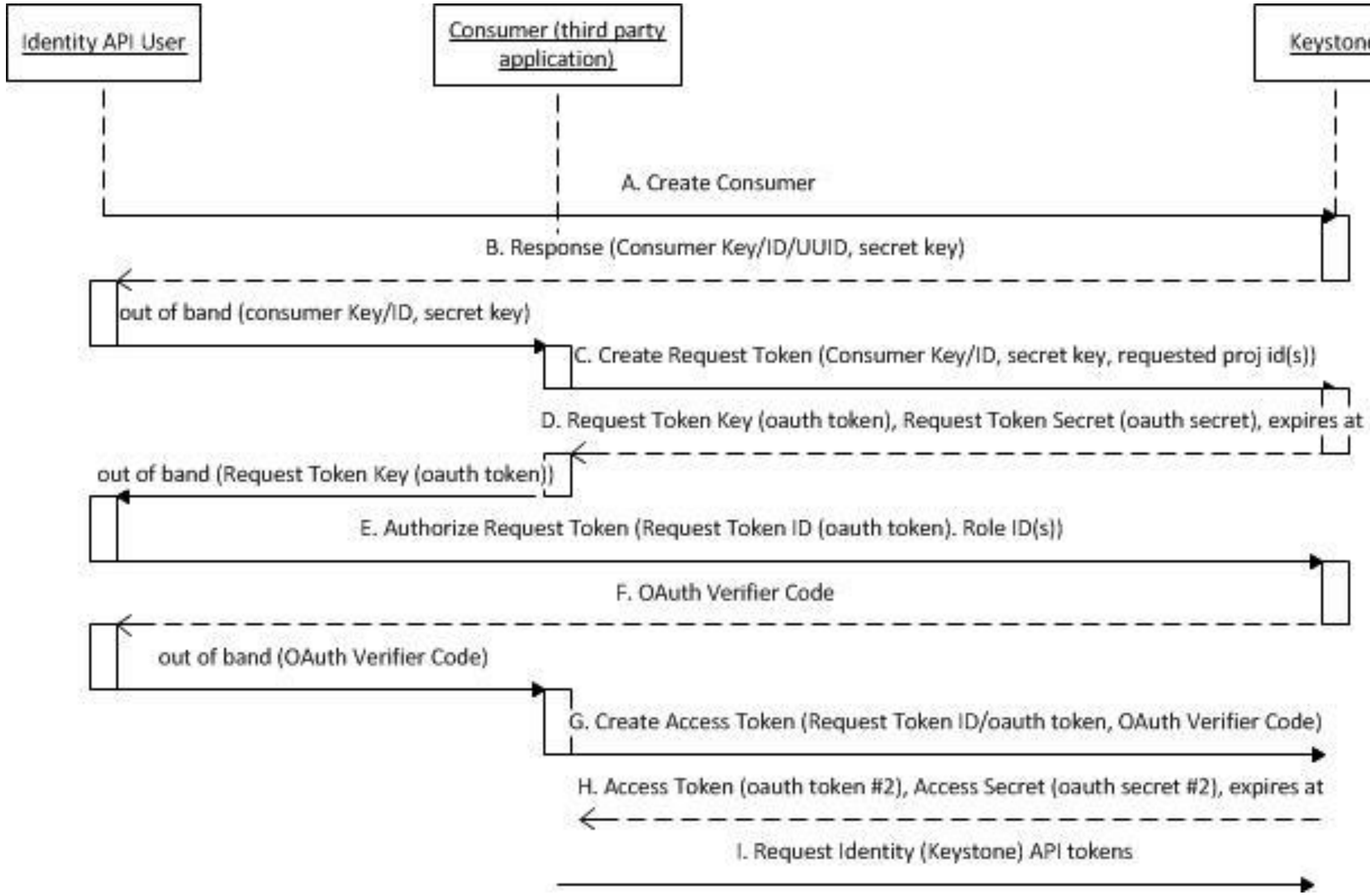
- To handle multi-node Keystone deployments:
  - After initial fernet_setup, distribute the fernet key repository
  - After each key rotate, distribute the fernet key repository

Fernet Tokens (cont’d)--Oauth

Oauth (Open Standard for Authorization) is a way for a user to employ their Google, Facebook, Twitter, etc. accounts to log onto a third party website

- so that doesn’t show user’s password to the third party web site

Fernet Tokens (cont'd)—Oauth (cont'd)


Identity API User
Consumer (third party application)
Keystone
A. Create Consumer
B. Response (Consumer Key/ID/UUID, secret key)
out of band (consumer Key/ID, secret key)
C. Create Request Token (Consumer Key/ID, secret key, requested proj id(s))
D. Request Token Key (oauth token), Request Token Secret (oauth secret), expires at
out of band (Request Token Key (oauth token))
E. Authorize Request Token (Request Token ID (oauth token). Role ID(s))
F. OAuth Verifier Code
out of band (OAuth Verifier Code)
G. Create Access Token (Request Token ID/oauth token, OAuth Verifier Code)
H. Access Token (oauth token #2), Access Secret (oauth secret #2), expires at
I. Request Identity (Keystone) API tokens

Keystone Trusts

- A user called a “trustor” can delegate a “trust” consisting of specified rights (one or more “Roles”) to a user called “trustee”
- A “trust” is an object
- A token can have a trust associated with it

Keystone Trusts (cont'd)

Trust Object Information used to create a trust (in JSON format)	
impersonation	allows the trustee to impersonate the "trustor"
project_id	identifies project for which the trust was delegated
expires_at	time at which the trust expires
remaining_uses	number of times a trust may be exercised
trustor_user_id	user who created the trust (trustor)
trustee_user_id	user who receives the trust (trustee)
roles	list of roles to be granted to the trustee

Keystone Trusts (cont’d)--API

- How Trust Interface Works:
  1. User A requests a token from Keystone:
     - Send HTTP Get to http://localhost:355357/v3/auth/tokens
  2. User A sends a POST to Keystone to create a trust at the /OS-TRUST/trusts interface
     - includes trust object in JSON format
     - includes User A token originally received from Keystone
  3. Keystone trusts interface responds
     - response includes a trust identifier for the trust that was created
  4. User A gives trust identifier to User B
  5. User B requests a token from Keystone
  6. User B performs a request for a Trust token
     1. Includes User A’s token
     2. Includes trust identifier