Multisite Admin User Guide

Release brahmaputra.1.0 (e122da8)

OPNFV

February 25, 2016

CONTENTS

CHAPTER

ONE

MULTISITE ADMIN USER GUIDE

1.1 Multisite identity service management

1.1.1 Goal

A user should, using a single authentication point be able to manage virtual resources spread over multiple OpenStack regions.

1.1.2 Token Format

There are 3 types of token format supported by OpenStack KeyStone

- **UUID**
- **PKI/PKIZ**
- **FERNET**

It's very important to understand these token format before we begin the mutltisite identity service management. Please refer to the OpenStack official site for the identity management. http://docs.openstack.org/admin-guide-cloud/identity_management.html

1.1.3 Key consideration in multisite scenario

A user is provided with a single authentication URL to the Identity (Keystone) service. Using that URL, the user authenticates with Keystone by requesting a token typically using username/password credentials. Keystone server validates the credentials, possibly with an external LDAP/AD server and returns a token to the user. The user sends a request to a service in a selected region including the token. Now the service in the region, say Nova needs to validate the token. The service uses its configured keystone endpoint and service credentials to request token validation from Keystone. After the token is validated by KeyStone, the user is authorized to use the service.

The key considerations for token validation in multisite scenario are:

- Site level failure: impact on authN and authZ shoulde be as minimal as possible
- Scalable: as more and more sites added, no bottleneck in token validation
- Amount of inter region traffic: should be kept as little as possible

Hence, Keystone token validation should preferably be done in the same region as the service itself.

The challenge to distribute KeyStone service into each region is the KeyStone backend. Different token format has different data persisted in the backend.

- UUID: UUID tokens have a fixed size. Tokens are persistently stored and create a lot of database traffic, the persistence of token is for the revoke purpose. UUID tokens are validated online by Keystone, call to service will request keystone for token validation. Keystone can become a bottleneck in a large system. Due to this, UUID token type is not suitable for use in multi region clouds, no matter the Keystone database replicates or not.
- PKI: Tokens are non persistent cryptographic based tokens and validated offline (not by the Keystone service) by Keystone middleware which is part of other services such as Nova. Since PKI tokens include endpoint for all services in all regions, the token size can become big. There are several ways to reduce the token size such as no catalog policy, endpoint filter to make a project binding with limited endpoints, and compressed PKI token - PKIZ, but the size of token is still unpredictable, making it difficult to manage. If catalog is not applied, that means the user can access all regions, in some scenario, it's not allowed to do like this. Centralized Keystone with PKI token to reduce inter region backend synchronization traffic. PKI tokens do produce Keystone traffic for revocation lists.
- Fernet: Tokens are non persistent cryptographic based tokens and validated online by the Keystone service. Fernet tokens are more lightweight than PKI tokens and have a fixed size. Fernet tokens require Keystone deployed in a distributed manner, again to avoid inter region traffic. The data synchronization cost for the Keystone backend is smaller due to the non- persisted token.

Cryptographic tokens bring new (compared to UUID tokens) issues/use-cases like key rotation, certificate revocation. Key management is out of scope for this use case.

1.1.4 Database deployment as the backend for KeyStone service

Database replication:

- Master/slave asynchronous: supported by the database server itself (mysql/mariadb etc), works over WAN, it's more scalable. But only master will provide write functionality, domain/project/role provisioning.
- Multi master synchronous: Galera(others like percona), not so scalable, for multi-master writing, and need more parameter tunning for WAN latency.It can provide the capability for limited multi-sites multi-write function for distributed KeyStone service.
- Symmetrical/asymmetrical: data replicated to all regions or a subset, in the latter case it means some regions needs to access Keystone in another region.

Database server sharing: In an OpenStack controller, normally many databases from different services are provided from the same database server instance. For HA reasons, the database server is usually synchronously replicated to a few other nodes (controllers) to form a cluster. Note that _all_ database are replicated in this case, for example when Galera sync repl is used.

Only the Keystone database can be replicated to other sites. Replicating databases for other services will cause those services to get of out sync and malfunction.

Since only the Keystone database is to be sync or replicated to another region/site, it's better to deploy Keystone database into its own database server with extra networking requirement, cluster or replication configuration. How to support this by installer is out of scope.

The database server can be shared when async master/slave replication is used, if global transaction identifiers GTID is enabled.

1.1.5 Deployment options

Distributed KeyStone service with PKI token

Deploy KeyStone service in two sites with database replication. If site level failure impact is not considered, then KeyStone service can only be deployed into one site.

The PKI token has one great advantage is that the token validation can be done locally, without sending token validation request to KeyStone server. The drawback of PKI token is the endpoint list size in the token. If a project will be only spread in very limited site number(region number), then we can use the endpoint filter to reduce the token size, make it workable even a lot of sites in the cloud. KeyStone middleware(which is co-located in the service like Nova-API/xxx-API) will have to send the request to the KeyStone server frequently for the revoke-list, in order to reject some malicious API request, for example, a user has to be deactivated, but use an old token to access OpenStack service.

For this option, needs to leverage database replication to provide KeyStone Active-Active mode across sites to reduce the impact of site failure. And the revoke-list request is very frequently asked, so the performance of the KeyStone server needs also to be taken care.

Site level keystone load balance is required to provide site level redundancy, otherwise the KeyStone middleware will not switch request to the healthy KeyStone server in time.

And also the cert distribution/revoke to each site / API server for token validation is required.

This option can be used for some scenario where there are very limited sites, especially if each project only spreads into limited sites (regions).

Distributed KeyStone service with Fernet token

Fernet token is a very new format, and just introduced recently,the biggest gain for this token format is :1) lightweight, size is small to be carried in the API request, not like PKI token(as the sites increased, the endpoint-list will grows and the token size is too long to carry in the API request) 2) no token persistence, this also make the DB not changed too much and with light weight data size (just project, Role, domain, endpoint etc). The drawback for the Fernet token is that token has to be validated by KeyStone for each API request.

This makes that the DB of KeyStone can work as a cluster in multisite (for example, using MySQL galera cluster). That means install KeyStone API server in each site, but share the same the backend DB cluster.Because the DB cluster will synchronize data in real time to multisite, all KeyStone server can see the same data.

Because each site with KeyStone installed, and all data kept same, therefore all token validation could be done locally in the same site.

The challenge for this solution is how many sites the DB cluster can support. Question is aksed to MySQL galera developers, their answer is that no number/distance/network latency limitation in the code. But in the practice, they have seen a case to use MySQL cluster in 5 data centers, each data centers with 3 nodes.

This solution will be very good for limited sites which the DB cluster can cover very well.

Distributed KeyStone service with Fernet token + Async replication (star-mode)

One master KeyStone cluster with Fernet token in two sites (for site level high availability purpose), other sites will be installed with at least 2 slave nodes where the node is configured with DB async replication from the master cluster members, and one slave's mater node in site1, another slave's master node in site 2.

Only the master cluster nodes are allowed to write, other slave nodes waiting for replication from the master cluster member(very little delay).

Pros:

- Deploy database cluster in the master sites is to provide more master nodes, in order to provide more slaves could be done with async. replication in parallel. Two sites for the master cluster is to provide higher reliability (site level) for writing request, but reduce the maintaince challenge at the same time by limiting the cluster spreading over too many sites.
- Multi-slaves in other sites is because of the slave has no knowledge of other slaves, so easy to manage multi-slaves in one site than a cluster, and multi-slaves work independently but provide multi-instance redundancy(like a cluster, but independent).

Cons:

- Need to be aware of the chanllenge of key distribution and rotation for Fernet token.

1.2 Multisite VNF Geo site disaster recovery

1.2.1 Goal

A VNF (telecom application) should, be able to restore in another site for catastrophic failures happened.

1.2.2 Key consideration in multisite scenario

Geo site disaster recovery is to deal with more catastrophic failures (flood, earthquake, propagating software fault), and that loss of calls, or even temporary loss of service, is acceptable. It is also seems more common to accept/expect manual / administrator intervene into drive the process, not least because you don't want to trigger the transfer by mistake.

In terms of coordination/replication or backup/restore between geographic sites, discussion often (but not always) seems to focus on limited application level data/config replication, as opposed to replication backup/restore between of cloud infrastructure between different sites.

And finally, the lack of a requirement to do fast media transfer (without resignalling) generally removes the need for special networking behavior, with slower DNS-style redirection being acceptable.

Here is more concerns about cloud infrastructure level capability to support VNF geo site disaster recovery

1.2.3 Option1, Consistency application backup

The disater recovery process will work like this:

1. DR(Geo site disaster recovery)software get the volumes for each VM in the VNF from Nova
2. DR software call Nova quiesce API to quarantee quiecing VMs in desired order
3. DR software takes snapshots of these volumes in Cinder (NOTE: Because storage often provides fast snapshot, so the duration between quiece and unquiece is a short interval)
4. DR software call Nova unquiece API to unquiece VMs of the VNF in reverse order
5. DR software create volumes from the snapshots just taken in Cinder
6. DR software create backup (incremental) for these volumes to remote backup storage (swift or ceph, or..) in Cinder
7. If this site failed,

1. DR software restore these backup volumes in remote Cinder in the backup site.
2. DR software boot VMs from bootable volumes from the remote Cinder in the backup site and attach the regarding data volumes.

Note: It's up to the DR policy and VNF character how to use the API. Some VNF may allow the standby of the VNF or member of the cluster to do quiece/unquiece to avoid interfering the service provided by the VNF. Some other VNF may afford short unavailable for DR purpose.

This option provides application level consistency disaster recovery. This feature is WIP in OpenStack Mitaka release, and will be avaialle in next OPNFV release.

1.2.4 Option2, Vitrual Machine Snapshot

1. DR software create VM snapshot in Nova
2. Nova quiece the VM internally (NOTE: The upper level application or DR software should take care of avoiding infra level outage induced VNF outage)
3. Nova create image in Glance
4. Nova create a snapshot of the VM, including volumes

5) If the VM is volume backed VM, then create volume snapshot in Cinder 5) No image uploaded to glance, but add the snapshot in the meta data of the

   image in Glance

6. DR software to get the snapshot information from the Glance

7) DR software create volumes from these snapshots 9) DR software create backup (incremental) for these volumes to backup storage

   (swift or ceph, or..) in Cinder

10. If this site failed,

1. DR software restore these backup volumes to Cinder in the backup site.
2. DR software boot vm from bootable volume from Cinder in the backup site and attach the data volumes.

This option only provides single VM level consistency disaster recovery.

This feature is already available in current OPNFV release.

1.2.5 Option3, Consistency volume replication

1. DR software creates datastore (Block/Cinder, Object/Swift, App Custom storage) with replication enabled at the relevant scope, for use to selectively backup/replicate desire data to GR backup site
2. DR software get the reference of storage in the remote site storage
3. If primary site failed,

1. DR software managing recovery in backup site gets references to relevant storage and passes to new software instances
2. Software attaches (or has attached) replicated storage, in the case of volumes promoting to writable.

Pros:

- Replication will be done in the storage level automatically, no need to create backup regularly, for example, daily.
- Application selection of limited amount of data to replicate reduces risk of replicating failed state and generates less overhear.
- Type of replication and model (active/backup, active/active, etc) can be tailored to application needs

Cons:

- Applications need to be designed with support in mind, including both selection of data to be replicated and consideration of consistency
- “Standard” support in Openstack for Disaster Recovery currently fairly limited, though active work in this area.

This feature is in discussion in OpenStack Mitaka release, and hopefully will be avaialle in next OPNFV release.

1.3 VNF high availability across VIM

1.3.1 Goal

A VNF (telecom application) should, be able to realize high availability deloyment across OpenStack instances.

1.3.2 Key consideration in multisite scenario

Most of telecom applications have already been designed as Active-Standby/Active-Active/N-Way to achieve high availability (99.999%, corresponds to 5.26 minutes of unplanned downtime in a year), typically state replication or heart beat between Active-Active/Active-Active/N-Way (directly or via replicated database services, or via private designed message format) are required.

We have to accept the currently limited availability (99.99%) of a given OpenStack instance, and intend to provide the availability of the telecom application by spreading its function across multiple OpenStack instances.To help with this, many people appear willing to provide multiple "independent" OpenStack instances in a single geographic site, with special networking (L2/L3) between clouds in that physical site.

The telecom application often has different networking plane for different purpose:

1. external network plane: using for communication with other telecom application.
2. components inter-communication plane: one VNF often consisted of several components, this plane is designed for components inter-communication with each other
3. backup plane: this plane is used for the heart beat or state replication between the component's active/standby or active/active or N-way cluster.
4. management plane: this plane is mainly for the management purpose, like configuration

Generally these planes are separated with each other. And for legacy telecom application, each internal plane will have its fixed or flexible IP addressing plane. There are some interesting/hard requirements on the networking (L2/L3) between OpenStack instances, at lease the backup plane across different OpenStack instances:

1. Overlay L2 networking or shared L2 provider networks as the backup plane for heartbeat or state replication. Overlay L2 network is preferred, the reason is:

   a) Support legacy compatibility: Some telecom app with built-in internal L2 network, for easy to move these app to virtualized telecom application, it would be better to provide L2 network.

   (a) Support IP overlapping: multiple telecom applications may have overlapping IP address for cross OpenStack instance networking Therefore, over L2 networking across Neutron feature is required in OpenStack.
2. L3 networking cross OpenStack instance for heartbeat or state replication. For L3 networking, we can leverage the floating IP provided in current Neutron, so no new feature requirement to OpenStack.

Overlay L2 networking across OpenStack instances is in discussion with Neutron community.