VOSS-4-UC
Best Practices Guide

Release 20.1.1 **Early Field Trial**

Jan 28, 2021

Legal Information

Please take careful note of the following legal notices:

- VOSS does not guarantee that this document is technically correct, complete, or that the product is free from minor flaws. VOSS endeavors to ensure that the information contained in this document is correct, whilst every effort is made to ensure the accuracy of such information, VOSS accepts no liability for any loss (however caused) sustained as a result of any error or omission in the same.
- This document is used entirely at the users own risk. VOSS cannot be held responsible or liable for any damage to property, loss of income, and or business disruption arising from the use of this document.
- The product capabilities described in this document and the actual capabilities of the product provided by VOSS are subject to change without notice.
- VOSS reserves the right to publish corrections to this document whenever VOSS deems it necessary.
- All vendor/product names mentioned in this document are registered trademarks and belong to their respective owners. VOSS does not own, nor is related to, these products and vendors. These terms have been included to showcase the potential of the VOSS solution and to simplify the deployment of these products with VOSS should you select to utilize them.

Security Information

This product may contain cryptographic features that may be subject to state and local country laws that govern the import, export, transfer and use of such features. The provision of this software does not imply that third-party authorization to import, export, distribute or use encryption in your particular region has been obtained. By using this product, you agree to comply with all applicable laws and regulations within your region of operation. If you require further assistance, please contact your dedicated VOSS support person.

Contents

1 Deployment Models and Web Weight Settings

1.1. Overview

- The supported deployment models are described in the Install Guide under Chapter 2 Deployment Topologies.
- Web weights are explained in the Install Guide under Multi Data Center Deployments and Multinode Installation. The web weight specifies the routing and relative counts of the initial HTTP request from the Web Proxy to a Unified Node. The initial request could be a request such as a transaction, or for example a GET request.
- Transactions can be processed on *any* Active Unified Node - regardless of which Unified Node processed the initial HTTP request. The transaction log provides the detailed information in the fields shown below:
  - `submitter_host_name`: the hostname of the application node that scheduled the transaction.
  - `processor_host_name`: the hostname of the application node that processed the transaction (this value is set once the transaction is processed).
- Note that a sub-transaction can be processed on a different Unified Node than its immediate precedent in the hierarchy.
- We recommend that you use both Web Proxies. However, the use of only 1 Web Proxy is supported (and Web Proxy use is optional).
- To display the configured web weights, run the command **web weight list** at the CLI of each Web Proxy Node .
- The recommended web weight settings for the various deployment models are shown in the following sections.

1.2. Active-Active Web Weights

- There are 4 Active Unified Nodes, 2 in each Data Center. The maximum supported Round Trip Time (RTT) is 10ms.
- WP1: 1 1 0 0
- WP2: 0 0 1 1

This scheme is designed to route the initial HTTP request to the Unified Nodes local to the Web Proxy Node that forwards the request. If only one Web Proxy (WP) is used, then use the following setting for WP1:

WP1: 1 1 1 1

This results in some of the initial HTTP requests crossing to the Secondary Data Center, however this has a minimal impact on system performance.

1.3. Active-StandBy Web Weights

- There are 4 Active Unified Nodes in the Primary Data Center and 2 StandBy Unified Nodes in the secondary Data Center. The maximum supported RTT is 400ms.
- WP1: 1 1 1 1 0 0
- WP2: 1 1 1 1 0 0

If only Web Proxy 1 (WP1) is used, the default weights provided by the system are sufficient. If Web Proxy 2 or both Web Proxy Nodes are used, change the web weights at Web Proxy 2 to the values noted above.

The logic behind the web weight settings for the Active-StandBy model is that some non-transaction work may generate a significant number of queries back to the Primary DB. Therefore, processing such work in the secondary Data Center may result in unacceptably long processing times.

2 Overload Controls

2.1. Session Limits

The numbers below represent the default and maximum values.

- global administration: 200 (includes non-customer admins as well as service provider and reseller admins). This limit also includes API clients configured as Admin Users.
- global self-service: 20,000 - This is the total number of self-service users logged into the system - both active and inactive.
- per customer administration: 10 (this should be set to a lower value in some cases). The Partner must first “reserve” a number of non-customer admin sessions from the global limit of 200 noted above. The remaining admins can be allocated to customers. Based on the expected number of customers, the Partner can then set the per customer admin limit.

  For example, if the Partner wishes to “reserve” 20 admin sessions for non-customer use, that would leave 180 available for customer use. If a total of 40 customers is expected, the Partner should set the per customer admin limit to no more than 180/40 = 4 (rounded down from 4.5). In this example, a maximum of 40 x 4 = 160 admin sessions can be allocated to customer-level admins. This would effectively reserve 200-160 = 40 admins for the Partner to use.
- per customer self-service: 1000

2.2. Throttle Limits

- Admin (by default, this is disabled). We recommend that the Admin throttle is enabled and set to 450 API requests/min. The setting is per Unified Node.
  - Service Inventory (SI) Report: Relies on the per-user throttle setting to ensure adequate throughput.
    - For the Active-StandBy deployment model, we highly recommended that the SI report is configured to run on a specific non-Primary Unified Node (preferably in the Secondary Data Center as those nodes are likely to have a lower load). This results in faster performance, but there is not any protection against a single node failure in the middle of an SI report run (not very likely). The use of a Web Proxy is *not* recommended here as 25% of the initial requests are routed to the Primary Unified Node based on the recommended web weight settings at either Web Proxy.
    - For the Active-Active deployment model, you can use a Web Proxy instead. The SI report would run slower, but this configuration provides protection against a single Unified Node failure during the SI report run. If a Web Proxy is used, then:

    - Use Web Proxy 2. This prevents routing to the Primary Unified Node based on the recommended web weight settings.
    - The Web Proxy knows the health of the UN and can route requests accordingly.

- Per User throttle for API Clients:
  - Default setting is 20 API req/sec per Active Unified Node. 4 Active Unified Nodes x 20 = 80 API req/sec (system wide) or 4800 API req/min (system wide). We recommend that you keep this setting.
  - This limit applies to all admin users, but in practice serves to limit API clients. Human admin users are not likely to create a traffic rate of 80 API req/sec.
- Self-Service throttle. The default setting is 300 API req/min (per Unified Node). APIs for logins and actions would count against this throttle.

3 Onboarding Customers and Users

3.1. Guidance on Planning for Onboarding and Ongoing Operations

This is a high-level view:

- Number of Parallel operations, for example BL and QAS, for best performance:
  - BL: 4x500 (4 Bulk Load sheets in parallel with a maximum of 500 rows per sheet).
  - QAS: 5x200 (5 QAS Bulk Load sheets in parallel with a maximum of 200 rows per sheet).
- Recommendations for sync operations:
  - We recommend that you schedule sync operations during off-peak hours.
  - During business hours, sync operations are slower due to the presence of other work on the system.
- Scheduler Template settings (20%, 50%, 80%):
  - For periods of high self-service and administrative work (including API clients), we recommend that the template is set to 20%.
  - For periods of moderate self-service and administrative work (including API clients), we recommend that the template is set to 50%. This is the value in the system prior to SU-1.
  - For periods of low self-service and administrative work (including API clients), set the template to 80%.
  - The current implementation only allows you to set 2 of the 3 values, that is a peak and an off-peak setting.
- In some cases, there are situations where VOSS-4-UC is used for changes. Ad Hoc syncs are best in this case.
- AS may choose to change CUCM and sync back to VOSS-4-UC. This is where you must schedule daily off-peak sync operations.

4 Data Sync

4.1. General Sync Principles and Best Practices

4.1.1. Sync Overview

VOSS-4-UC provides a number of features for the system to stay in sync with the underlying UC applications, thereby allowing for the configuration and management of the UC apps outside of VOSS-4-UC when required.

- Cache control policy - this mechanism in the VOSS-4-UC system provides the ability to pull in the latest live data from the UC application(s) for the entity that you are viewing or at the time that it is needed, for example before executing a change on that entity; to prevent any overwrite or setting conflict.

  For more details on the cache control policy behavior and configuration, see the Data Sync chapter in the Core Feature Guide.

- Data Sync - This is a workflow that will pull the latest data from the UC apps and update the VOSS-4-UC cache when ran adhoc or via a schedule. This is typically used for processes like overbuild to pull in the existing configuration from the UC applications or to pull in other changes made in the UC apps outside of VOSS-4-UC.

  For more information on the sync behavior and configuration, see the Data Sync chapter in the Core Feature Guide.

- With the cache control policy in place, the need to setup and schedule sync regular syncs should aim to address any gaps that the cache control policy will not handle. Some of the prime use cases and guidelines on when a regular or scheduled sync might be required for an entity versus the use of the cache control policy are as follows (these all assume some level of regular configuration being done to the UC apps directly outside of VOSS-4-UC):

  – If you are adding/removing entities (e.g users, phones, etc) in the UC application(s) directly, then a sync is required to pull in new entities or remove existing entities.

  – If you are modifying key values that appear in the list views in VOSS-4-UC via the applications directly (e.g changing a user's name), then an update sync might be required. The list view data is driven only from the VOSS-4-UC cache so any updates made in the UC apps will not be shown in VOSS-4-UC until the entity is viewed in VOSS-4-UC (for example, opening that subscriber) or when an update sync is run.

  – Any type of extract that might be run from VOSS-4-UC (file dump, VOSS-4-UC Analytics, billing feed, etc) would be based on the cached data in VOSS-4-UC. So a sync may be required if those capabilities are in use and any of the critical settings in those extracts are being managed outside of VOSS-4-UC.

  – External clients accessing VOSS-4-UC via the API have the cached flag available to request VOSS-4-UC cached data (`cached = true`) or to have VOSS-4-UC retrieve the latest data from the UC apps before responding (`cached = false`). So the presence of this external client does not require a regular sync to be run as it can (and likely should) request the latest data in any case depending on the use case.

  - Any other mods (e.g call forward on a line via the CFwdALL softkey) made on an entity will be pulled in when the record is viewed so do not necessarily require a sync.

For example, if the only concern is that when executing an update to an entity that the latest current settings are shown, then the cache control policy handles this without the need for a regularly scheduled update sync.

When the sync is run (manual or via schedule), the hierarchy that the sync is run on will determine where the items are pulled into. For example, a sync at the customer level will pull data in at customer level, while a sync at a site will pull data in at the site.

So when setting up the sync, consider the purpose: if the items being pulled in need to be in a site, it might be more efficient to set up the sync at that level and run it there, as opposed to syncing in at the customer and then having the move the various elements. This can even be done as a once-off sync and with the use of model type lists and model instance filters to grab the data relevant for the site.

4.1.2. Data Sync Types

Data synchronization is available with the following functionality:

- Merge data by pulling data that is on the device but not in the cache, to the cache and vice versa
- Pull all data from the device
- Pull only the schema from the device (used for LDAP)
- Pull data from the Change Notification Feature local data collection
- Purge data from the cache
- Push data in the cache to the device

A quick import option is available to fetch only summary data that is contained in a list operation response and not the data for all instances/fields.

For all the syncs in general, the VOSS-4-UC system builds up the lists of entities from both VOSS-4-UC and the device (for example Unified CM) for comparison. The field that is used for this comparison is the key for the device entity, which is typically the unique identifier for the record in the device we that are syncing with. For example, for Unified CM, the identifier is the pkid which is the internal Unified CM database id.

In the case of subscribers, a sync will build up the list of `device/cucm/Users` in VOSS-4-UC and then request from the Unified CM the lists of users it currently has for the comparison. The differences in the lists are then handled according to each sync type.

Details of the sync types are listed below:

- Pull from Device - The VOSS-4-UC resource is updated where the same key is present in both lists. In this case, the device data is the master and the VOSS-4-UC system model data is updated with the device data.
  - For example, if new data is added to the Unified CM so that the VOSS-4-UC system data state for a Unified CM `device/cucm/User` does not show instances that are shown on the Unified CM, pull data synchronization synchronizes the system data with the Unified CM data. For example, a user's Department may be updated on the Unified CM, but this update will only show on the system after Pull from Device synchronization. If a user resource is created in Unified CM but not in VOSS-4-UC, this will add the `device/cucm/User` instance into VOSS-4-UC *at the level the pull sync was run from*, for example at the customer level.

  - In the case where you delete a VOSS-4-UC resource from the device so that the key is in the VOSS-4-UC list but not in the 'device' list, a pull sync will remove the resource in VOSS-4-UC. For example, if the resource is a user in VOSS-4-UC but not in Unified CM, the pull sync will remove the `device/cucm/User` record in VOSS-4-UC.

  If you are pulling device data, for example LDAP users from an LDAP device, the results returned to VOSS-4-UC are dependent on the LDAP server configuration. For example, if the returned results exceed the LDAP server configured maximum and if the server does not support paging, then an appropriate error message is returned.

- Push to Device - The VOSS-4-UC system data state is the master and devices are synchronized with it.
  - Where you delete device data from VOSS-4-UC so that the key is in the 'device' list but it is not in the VOSS-4-UC list, for example deleting a user in VOSS-4-UC, this will remove the user from Unified CM to match the user not existing in VOSS.
  - If new device data is added to the VOSS-4-UC system data so that the resource shows instances that are not shown on the device, push data synchronization synchronizes the device data with the VOSS-4-UC system data. For example, adding a `device/cucm/User` instance to VOSS-4-UC and running a Push to Device sync will add the user record to Unified CM.

  Keys found in both lists are ignored, for example no updates are made for existing records in either direction. So in the `device/cucm/User` example, if the same user exists in both the VOSS-4-UC and Unified CM system, then no update occurs in either direction. In other words, detailed settings may still not match after a Push to Device sync.

- Merge with Device - This is a combination of pull and push data synchronization and takes place in both directions.

  Merge data synchronization synchronizes data between the system and the Unified CM data without overwriting or deleting any data.
  - If you create a resource where the key is in the 'device' list but not in the VOSS-4-UC list (for example, the entity is in Unified CM but not in VOSS-4-UC), then the VOSS-4-UC resource is created. For example, adding a user in Unified CM will create the `device/cucm/User` record in VOSS-4-UC after the merge sync.
  - If you add to a device so that the key is in the VOSS-4-UC list but not the 'device' list (for example, the entity is in VOSS-4-UC but not in Unified CM), then the device resource is created. For example, creating a `device/cucm/User` record in VOSS-4-UC will create the user in Unified CM.

  Keys found in both lists are ignored, for example no updates will be made for existing records in either direction. So in the `device/cucm/User` example, if the same user exists in both the VOSS-4-UC and Unified CM system, then no update occurs in either direction. In other words, detailed settings may still not match after a Merge with Device sync.

Note: Bear in mind that with the above keys list sync logic, in the case of a reversion of the Unified CM to restores/inactive partitions, the relevant pkids might be different in the end state than they were at the last time VOSS-4-UC was in sync with Unified CM before the restore - especially if there was a lot of testing in between.

What this means practically is that there could for example be a user with the same username in both VOSS-4-UC and Unified CM, but if that user's pkid in Unified CM is now different to the one that VOSS-4-UC holds from previous syncs or interactions, then the users will be seen as different even though the usernames are the same.

- Change Notification Sync - A pull sync of changes stored in the local collection that is updated by the Change Notification Collector service.

  For more details on CNF, refer to the relevant CNF topics in the Data Sync chapter of the Core Feature Guide.

- Purge Local Resources - All resources or instances of device information that exist in the system will be deleted. However, the entities in the device will not be deleted. This option is typically used when cleaning up the system. Note that a warning will be displayed prior to executing an enabled Purge sync.

4.1.3. Key Settings for Data Sync

A number of key settings that are available for data sync:

- Model Type lists - define which entities to pull in a given sync (for example, only pull in `device/cucm/User` records from Cisco Unified CM).
- Model Instance filters - limit a sync to a subset of entities in a sync (for example, pull in users with a primary extension starting with 1). This setting requires a system level administrator to expose it on the Admin Portal.
- Actions - select which actions (Add/Update/Delete) are active for a sync. Update tends to be the most effort to run, because for most systems this involves a GET API call for each record and comparing to our data. Add/Del can be determined from the initial list API calls. If you are really only need the Add and/or Del action, then disabling the Update action will likely save considerable time on the sync.
- Quick Import - use the list API responses to update the VOSS-4-UC cache and will not do individual GET calls for each entity for the update. This works well where the list response contains all the values for the entity or only key settings need to be updated. The removal of all the individual GETs means the sync occurs much faster, because VOSS-4-UC is not waiting for the API responses when there are a many entities to update. This is useful if the list and GET responses are needed or if you only need the summary data from the list view.

Note: The Quick Import option is not recommended in most cases, but should only be used for the sync of `device/cuc/ImportUser`. However, *initially* there is an exception to the performance improvement of a quick import sync with `device/cuc/User`:

When quick import is turned on on a sync which has previously run without it, dependent, non-Import User model types use the LIST response data to compare with the resource data which was originally saved using the GET response data. The data sync detects a change and a resource save is initiated for each instance. In the case of `device/cuc/User`, this means that dependent import API calls are made, which result in a long sync time.

Once it has completed, however, *subsequent* quick import syncs should show an improvement over non-quick import syncs, but when changing back to a non-quick import sync, the same effect would likely be observed.

4.1.4. Scheduling Syncs

- When scheduling syncs, avoid too many overlapping syncs at a given time. VOSS-4-UC already blocks multiple syncs against a given device.
- The best practice is to not have more than 5 syncs running at a given time.

- To avoid load and issues with concurrency, schedule syncs carefully and at intervals when they are really required. For example, do not run nightly syncs unless it is necessary. Since syncs generally cover the case where information is changed on the UC apps outside VOSS-4-UC, the level of third party integration or direct configuration tasks should play a role in the decision to schedule. For details, refer to the topics on Cisco Unified CM, CUC and LDAP below.
- Since it is possible to limit the number of records processed with a given sync, more predictability can be obtained with scheduling.

4.2. Cisco Unified CM

4.2.1. Cisco Unified CM Sync

Cisco Unified CM supports two types of sync:

- Regular API sync - utilizing the regular use of LIST and GET API calls to retrieve data; like any other device sync.
- Change Notification Sync - utilizes a service on the Cisco Unified CM side to pull information about records that have changed in a given period. Note: Model Instance Filters cannot be used in conjunction with a Change Notification Sync.

  The Change Notification Sync type is generally the most efficient sync type to use, as it minimizes the amount of data that needs to be retrieved from the Cisco Unified CM (especially for updates).

  The Change Notification Sync process works as follows:

  VOSS-4-UC retrieves the change records from the Cisco Unified CM on a regular interval (configurable). For example, this could be every 10 minutes. At the time of a scheduled sync is setup, VOSS-4-UC processes the change records collected (for example, nightly). VOSS-4-UC then processes the records accordingly:

  – Add - will do a GET API call to retrieve the full record and add it to VOSS-4-UC.
  – Update - will do a GET API call to retrieve the full record and update the record in VOSS-4-UC.
  – Del - will remove the record from VOSS-4-UC.

  So the efficiency on update syncs is because we do not need to do a GET API call for every single record in the system - only those that changed. In large UC application installations, this can make a big difference in Update sync times.

4.2.2. Update Sync Operations

A new feature introduced in VOSS-4-UC 17.4 permits the partner to use the Change Notification Feature of the CUCM to process update sync operations faster. The feature is OFF by default, but can be turned on by the partner.

The information here provides guidelines for setting up a sync schedule and lists the associated performance implications. Details on the operation of this feature are provided in other documents such as the Core Feature Guide. The changes mentioned here are not transactions. As a result, information is not displayed in the translation log, but rather in the special logs created for this feature.

The guidelines presented here are derived from concepts related to total processing capacity. The total number of updates processed in a time period is the sum of all of the updates across the customers selected for update in that time period. In our case, the time period is one hour. In this example, we assume that each customer has 1000 CUCM-related changes in that hour. The recommendation noted in the table that follows indicates that 5 customers can run in parallel (concurrently), and therefore a total of 5,000 changes processed in total.

If the partner exceeds the recommendation of 5 concurrent customers, a performance degradation may be observed, and the full set of required changes may not complete within that hour. Alternatively, if the number of changes for any customer is significantly higher than the 1,000 or if the total number of changes is significantly greater than 5,000, then the concurrency number supported may be less than 5.

If some of the planned changes do not complete within the hour noted in the table below, then those changes are completed the next time that particular customer is scheduled for a sync. If the number of changes for any customer is so large that the changes continually exceed those that can be processed in one hour, it will eventually result in a full sync. For such customers, we advise to schedule within an hour where less than 5 customers execute concurrently.

Configuration	Recommendation
Maximum number of concurrent CNF syn	5
Maximum number of changes processed per CNF sync	1,000
CNF sync schedule frequency	Once per hour per customer - This is subject to the staggering of CNF sync across customers.
Staggering of CNF syncs across customers	Factor of maximum changes processed and maximum number of concurrent CNF syncs.
CNF collector frequency	Initial recommendation is 15 minutes.
When is Full sync required?	Weekends only or when there are CNF alerts prompting for full sync.

If you experience a significant performance issue, you can turn the feature OFF again. Contact your support representative if you have any performance concerns.

4.2.3. Staggering of CNF Syncs Across Customers

Below follows an example and considerations:

If a Partner has 20 customers who want to use CNF sync, only schedule a maximum of 5 CNF sync to run concurrently. This means that syncs would run as follows:

- 1st hour, for example 12:00

  Customer 1 to Customer 5

- 2nd hour, for example 13:00

  Customer 6 to Customer 10

- 3rd hour, for example 14:00

  Customer 11 to Customer 15

- 4th hour, for example 15:00

  Customer 16 to Customer 20

- 5th hour, for example 16:00 (Begin repeating customers)

  Customer 1 - Customer 5

- and so on.

The preceding example means that the CNF sync schedule per customer must run at 4 hour intervals. Therefore, there are 6 CNF syncs per customer within a 24 hour window. With each CNF sync processing up to 1k changes, there are:

- A total of 6k changes processed per customer in a 24 hour window
- A total of 120k changes processed across all 20 customers in a 24 hour window

4.2.4. Recommended CUCM Sync Setups

Cisco Unified CM (CUCM) sync recommendations are covered here.

Bottom Up User Sync

If using bottom up sync into CUCM, the users are added to CUCM via LDAP. In this scenario they do not appear in VOSS-4-UC in order to be managed until they are synced in.

Note: If you use this sync in a multi-cluster environment, additional guidance on the user sync setup is provided in the Multi-Cluster Deployments Technical Guide.

- Recommended setup
  - Model Type List - `device/cucm/User`
  - Actions - Add/Update/Del all enabled.
  - When to use - scheduled. The most frequent this should run is in line with the LDAP->CUCM sync time (typically once every 24hrs but minimum of every 6hrs or so). The length of this sync should consider the maximum allowable time for an end user to be in the system in typical business practices. Edge cases can always be handled in between scheduled syncs by running the sync manually if required - that is often better than having a very frequent sync that is not typically needed.
  - Events - the different actions (add/update/del) have different post execution events for the `device/cucm/User` model type that need to occur. These handle various aspects of the user setup. See below for a screenshot of the setup for an example:
    - Add Operation workflow = `UserCucmSyncAdd`
    - Update Operation workflow = `UserCucmSyncUpdate`
    - Delete Operation workflow = `UserCucmSyncRemove`
  - Change notification should be used for this sync to manage load (except if using a model instance filter).

Workflows fields of the event setup on the CUCM User sync:

- **Model Type**: `device/cucm/User`

  Operation: Add

  Phase: Post Execution

Workflow: `UserCucmSyncAdd`

- **Model Type**: `device/cucm/User`

  Operation: Update

  Phase: Post Execution

  Workflow: `UserCucmSyncUpdate`

- **Model Type**: `device/cucm/User`

  Operation: Delete

  Phase: Post Execution

  Workflow: `UserCucmSyncRemove`

Phone Types and Related Entities

This will force VOSS-4-UC to retrieve the latest phone type data from the CUCM and related entities like phone button templates, and so on. This is not possible via the change notification in CUCM today.

- Recommended setup:
  - Model type list including: *device/cucm/PhoneType*, *device/cucm/PhoneTemplate*, *device/cucm/securityProfiles*
  - Actions - Add/Update/Del all enabled
  - When to use - Not scheduled - run ad hoc as needed. This includes post CUCM upgrades, installation of a new device COP file in CUCM, managing phone button templates, managing device security profiles. If you are not seeing a phone type of the button template in VOSS-4-UC that you are expecting, running this sync will likely resolve it.

Other Syncs

Beyond the syncs above, others can be setup to suit specific needs based on the implementation.

Important: In setting up processes that sync any new entities into VOSS-4-UC, these will add the entities to the hierarchy level of the sync. So this will require the use of overbuild or ad hoc move processes to get the entities into the right site, for example, if needed (such as users, phones, lines, and so on).

4.3. LDAP

4.3.1. LDAP

The LDAP sync process currently only supports regular syncs.

4.4. Cisco Webex Teams (Spark)

4.4.1. Cisco Webex Teams Sync

If Cisco Webex Teams (Spark) is part of the solution and being managed, there are a number of considerations around sync with Cisco Webex Teams.

The typical setup is that the Cisco Webex Teams users are fully managed by the VOSS-4-UC system so there is no need for user sync. In this setup the only sync required is to pull in basic system data from Webex Teams for VOSS-4-UC to utilize in user configuration. A sync for this is added into VOSS-4-UC when a Cisco Webex Teams Service is added to the system and is executed automatically after Service creation or can be initiated by an admin as needed:

- SyncSparkRolesLicenses<customername> - Sync of basic data - e.g. licenses and roles, etc.

In an alternate scenario where some element of user management is occurring outside of VOSS-4-UC (for example, LDAP Connector), then a user sync will be required to pull that data into VOSS-4-UC for further management. Once the users are synced into VOSS-4-UC, they need to be moved to the appropriate site with the rest of the end user's services to be further configured and managed. This move can be done via the Webex Teams menu item by selecting the users and then using the **Action > Move** option to move them.

This sync can be initiated by an administrator as needed or if required, a schedule can be setup to run the sync on a regular interval.

- SyncSpark<customername> - Full sync of Webex Teams (Spark) including user data.

When VOSS-4-UC is integrated with a customer's user directory, the normal Subscriber management approach applies, in other words:

- users will synced into VOSS-4-UC at the Customer hierarchy level
- users must be moved to the relevant Site hierarchy level
- once at the correct Site level, Quick Add Subscriber or Advanced Subscriber can be used to enable services (Webex Teams in this case) for the users

5 Data Collection

5.1. Recommended RIS API Data Collector Interval

As a guideline to determine the interval that the (RIS) data collector service should poll the Unified CM, consider that:

- it takes about 14 minutes to collect information for around 200K phones on a cluster

The default value of the **RIS API data collector interval**: `43200` seconds (12 hours) can be adjusted accordingly.

Note: Collection processes run in parallel for each Unified CM on VOSS-4-UC.

To adjust the value, refer to the System Monitoring Configuration section in the Advanced Configuration Guide.

6 API Performance

6.1. API Resource Listing Best Practice

This section provides best practices when using API GET requests when listing resources. The best practices for the use of a number of API request parameters and parameter values are examined.

For further details on API parameters, refer to the API Guide.

The list of API request parameters for resource listing are:

Parameter	Description	Value	Default
`skip`	The list resource offset as a number.		0
`limit`	The maximum number of resources returned. The maximum value is 2000. If the `Range` request header is used, it will override this parameter.	1-2000	50
`count`	Specify if the number of resources should be counted. If false, the `pagination` object in the response shows the `total` as 0, so no total is calculated and the API performance is improved.	true, false	true
`order_by`	The summary attribute field to sort on.		First summary attribute
`direction`	The direction of the summary attribute field sort (asc:ascending, desc: descending).	asc, desc	asc
`summary`	Only summary data is returned in the data object.	true, false	true
`policy_name`	Return a model form schema where the Field Display Policy with name [FDP name] is applied to it. Use `policy` with the parameters `schema` and `format=json`.	[FDP name]	
`cached`	System will respond with resource information where the data was obtained from cache. (Functionally only applicable to device models and data models).	true, false	true
`api_version`	Return the the resource with `api_version`. Use with the parameter `format=json`	supported version no.	

Consider the following comments and best practices for the parameters below:

- `count`: The value of `count=true` is very expensive in terms of performance, and more so as the size of the resource grows. The first count query of for example a 36 000 Data Number Inventory resource can take as long as a minute to return a response. However, subsequent calls should decrease in execution time.

The value `count=true` should only be used if it is unavoidable. An alternative is to iterate over pages (`limit=200`) until the request returns less than 200 instances, or to simply paginate until no more resources are returned.

- `order_by`: no performance change if another summary attribute is specified.
- `direction`: no performance change if either values `asc` or `desc` are used.
- `policy_name`: the parameter is used by the GUI for display purposes. Timing data shows that the initial call with this parameters takes longer thank subsequent ones, possibly because of cache priming after a restart. Subsequent calls shows the execution time is on par with requests that do not include the parameter.
- `summary`: depending on the data required by the request, time can be saved if the value `summary=true`, so that only the summary data is returned.
- `limit`: execution time and memory consumption is impacted if the `limit` value is large.

To summarize, the recommended parameter values for an optimal API list request (GET) are:

- `cached=true`
- `summary=true`
- `count=false`
- `policy_name` not used

Example results with various parameter values (36 000 Data Number Inventory resource):

```
count:true, skip:0, policy_name:, limit:200, summary:false in 6.51744103432 s
count:true, skip:0, policy_name:, limit:200, summary:true in 5.6118888855 s
count:false, skip:0, policy_name:, limit:200, summary:false in 1.55350899696 s
count:false, skip:0, policy_name:policy_name=HcsDNInventoryDatFDP, limit:200,
     summary:true in 5.17663216591 s
count:false, skip:0, policy_name:, limit:200, summary:true in 1.09510588646 s
```

6.2. Long Running API Requests

To optimise memory utilization and performance, the system has been configured so that the API server will manage workers with the following defaults:

- after receiving a restart signal, workers have 100 minutes to finish serving requests
- a random restart interval of between 0 and 600 requests per worker (4 workers per node, 4 nodes in a cluster)

API best practices is to schedule and then poll transactions, since long running requests can affect recycling. In other words, preferably short requests and then poll.

6.2.1. Polling Example

To retrieve the status of a given transaction:

```
GET /api/tool/Transaction/[pkid]/poll/?format=json
```

The response contains essential status of the transaction, for example:

```
{
    [pkid]: {
        status: "Success",
        href: "/api/tool/Transaction/[pkid]",
        description: "Name:RDP-auser1857 Description:RD for auser1857"
    }
}
```

Refer to the topics *Poll Transactions* and *Example of an Asynchronous Mutator Transaction with nowait=true* in the API Guide.

7 System Maintenance

7.1. Transaction Archiving

The following are considerations when determining the frequency of the transaction archiving schedule to set up on the system. If a schedule is not set up for transaction archiving, system Alerts will be raised as well as a warning on the platform CLI login:

```
TRANSACTION DATABASE MAINTENANCE NOT SCHEDULED
```

- Run **voss transaction count <days>** on your system to inspect the number of transactions during a given period to determine your usage metrics.

  Refer to the *Database Commands for Transaction Management* topic in the Platform Guide for details on transaction archive command use and scheduling:

  - **voss transaction delete <days>**
  - **voss transaction export <days>**
  - **voss transaction archive <days>**

- Business policies - company policies may drive your choices: the immediate access to transaction logs for a period of time, security policy on data/audit retention, and so on.

  Note: The transaction archive process does mean the logs are not lost, just that they are not immediately accessible in the administrator graphical interface for searching.

- You can also set up system monitoring thresholds so that you receive alerts via the GUI and SNMP if the threshold is exceeded - which might indicate you need to review the archive schedule to increase how frequently is runs.

  See the *SNMP* and *VOSS-4-UC System Monitoring Traps* topics in the Platform Guide.

7.2. Automated Database Cache Cleanup

From VOSS-4-UC release 19.3.2 onwards, it is now longer necessary to schedule or manually manage the database cache optimization using the **voss trim-cache** command.

From release 20.1.1, this command is no longer available. A resource history is now maintained as a series of resource differences and is automatically optimized.

Note: A minimum retention period of 7 days is applied to resource differences in the resource history.

8 Administration Portal Setup

8.1. Navigation - Menu and Landing pages

The system provides a lot of tools to tailor the portal experience to your specific needs.

The advanced admin portal utilizes two key ways to provide users with the means to navigate around the system to key capabilities:

- Menu - left hand side of the screen and fully configurable
- Landing Page - page seen upon login or after using the **Home** link - also fully configurable

Both the menu and the landing page are fully configurable and this provides the key capabilities that are needed to create simple and efficient navigation experiences:

- What the entry in the menu/landing page is called - this should use terminology and reflect the tasks the user needs to do. This can be the business process for simpler admins or more technical terms for the advanced users.
- What the menu item links to - this is typically the form/view, listing, or other model in the system they need to access for that task.
- Display Policy - for views or when they select an entity from a list view, this determines the form layout they will see.
- Configuration template - this is applied when a view is accessed from the menu item, or the add action is selected from a list view. This can drive the entity during the add process. This can also be for visible fields to act as a default (or a fixed value if the field is read-only) or drive fields hidden by the display policy to provide fixed values.
- Filtering - both mechanisms provide advanced filtering mechanisms to drive different experiences. There are two key types of filters:
  - Fixed Filter - this is defined on the menu and cannot be seen/changed/removed by the user. The user is unaware a filter is applied and it is the baseline list view they see. They can apply further filtering as needed.
  - Configurable Filters - these are filters that can be fully or partially defined in the menu for items pointing to a list. This option gives an interim step between clicking on the menu/landing page option and getting to a list view. It will pop-up the filter options for the user to enter any filter criteria they require and it will be pre-populated with any criteria defined in the menu. Once submitted the list view is rendered using the provided filter criteria. The filter can be seen, changed, or removed as needed by the user.

See the core feature guide for more details on each of the elements and the detailed settings for each. Below we'll outline some suggested strategies and considerations to utilize to create the most efficient means for you different users types to get to the key capabilities they need.

The general strategy is that you want to make the most common tasks as quick and easy to get to for the different user types of the system. We suggest you use these capabilities to create the menu and landing page experiences you need to suit the different user roles that you set up in the system. In addition, the capabilities and experience needed for the user roles should be reviewed regularly with the users to look for additional opportunities to streamline and improve their experience to drive even greater efficiency or evolve to their changing needs.

Here are some key goals that should be provided through these capabilities.

8.1.1. Quick Access to tasks/searches:

The landing page should be populated with the most common tasks and/or searches that the user will be performing. This gives one-click access to those tasks/search from a single place and they can always quickly return via the home link at the top of the page. This makes the experience far easier and more intuitive for the user and saves them from needing to learn a specific menu structure or where items are to access. It is front and center and in terms they can easily understand.

Some examples of this:

- The top MACDs they do in the system should be on their landing page and easily accessible - with appropriate display policies and configuration templates. See the Simplified and Streamlined feature experience section below for more guidance on this
- Create landing page entries that are saved searches with defined filter criteria for one-click access. This can save time and effort as well as make the searches available to a wider audience. As an example of this - List un-registered phones which would be a link to phones with the criteria set to status starts with Un-Registered. This would the user one click access to unregistered phones in the system.
- Create landing page entries that have some criteria defined to help guide the user through a search they frequently do but has varying criteria. This walks them through the search process rather than going to a list view, to them pull up a filter and define all the criteria each time. For example, find a phone by user - this can be done with a landing page entry for phone, filter criteria set to ownerid, and then the user will be prompted to provide the required username and submit. This would give them one click access to finding a phone when they have a userid to work with.
- If there are more tasks than landing page space, then the menu can provide that access to the more edge case and deeper functions that the user might need from time to time.

8.1.2. Simplified and Streamlined feature experience

Rather than create a single link for a feature that handles a lot of different scenarios, it can be better to include multiple menu/landing page entries to the same feature with different display policy, configuration template, and filter options tailored to a specific use case. This can be a very simple way to create an experience of the feature that better suits a specific use case resulting in a more intuitive experience and improve automation. This streamlined experience can also reduce errors or reliance on users to follow a procedure and enter the right information for different scenarios through the same feature. This can also allow what would typically be more advanced capabilities to be exposed to lower level admins in a way that aligns to the business function.

Some examples:

- Create a link for adding SIP trunks as part of a specific 3rd party application integration that is regularly added. This can link to the SIP Trunk device model, utilize a display policy that hides virtually all the settings except those require entry - such as IP address and port of the remote system. The configuration template could define all the other technical settings according to that scenario (e.g CSS, call presentation information, digit manipulation, etc).

- Creating lines for different scenarios - there are a lot of different lines settings and optimizing the experience for different scenarios can greatly reduce effort and errors in setup. The display policies can be used to cut the visible fields down to those strictly needed for entry, while the configuration template can drive many of the detailed settings for the different scenarios. The result is you could add menu items for lines type A, line type B, etc. This makes it very simple to create these different types that align to the business task they understand without potential errors of the user deciding the right settings for the situation. This can even be combined with the filtering capability in the menu/landing page to separate these line types in the listing for full separation.
- UCM feature management - for some UCM capabilities there is not a specific feature built in VOSS-4-UC for managing it however they can still be accessed via the device models directly. The default device model layout is driven by the API definition from Cisco and can often include field names and order that do not align to the admin experience. This can easily be improved with a display policy to create the order you want and field labels that suit your needs. These can also be combined with Configuration Templates again to set defaults or drive hidden values to simplify the input and reduce errors in setup.

Index