Integration Background

Microsoft SCCM is made up of several components, which you should familiarize yourself with. The majority of all SCCM information is located within the SCCM database. There are also client/server-side logs that also produce value, but these logs haven't been included in this sample version of the app. You'll likely want to leverage these for your app.

When this sample app was initially designed, the most important of information was located within the SQL Database. In order to access data in a relational database, the Splunk DB Connect app is your go-to choice.

DB Connect

DBConnect is a pre-requisite to leveraging this sample SCCM App.

To use the DB Connect App, you can access it at the following URL:
https://apps.splunk.com/app/958/

DB Connect has some pre-requites, which are listed in the documentation. Once installed, you'll configure a connection to the server and database via the Splunk web interface. This creates a stanza in a "database.conf" file located in:

/opt/splunk/etc/apps/dbx/local

OR

C:\Program Files\Splunk\etc\apps\dbx\local

While you can see the stanza in this file, you will notice that the password is hashed out – so in order to setup your connection it must be done through the web interface.

Once you've completed this, you can move to the next section.

Inputs

Under the SCCM App directory, you will find a folder named local. These settings override anything in the default directory. In most cases, for anything that might be environment-specific, you will always want to make changes to the file within the local directory. For more detail or App Best Practices, navigate to the following URL- http://foo.com

If you review the inputs.conf, you will notice stanzas that look like the following:

```
[dbmon-dump://sccm/sccm-services]
 index = sccm
 output.format = kv
 output.timestamp = 0
 query = select * from Services_DATA
 sourcetype = sccm-services
 interval = 5
 table = sccm-services
```

OR

```
[dbmon-tail://sccm/sccm-services-tail]
index = sccm
interval = 5
```

output.format = kv
output.timestamp = 1
output.timestamp.column = TimeKey
output.timestamp.format = yyyy-MM-dd HH:mm:ss.SSS
sourcetype = sourcetype=deleteme
table = Services_DATA
tail.rising.column = TimeKey

The approach behind this is largely dependent upon whether you're doing a full table dump or a tail (both on scheduled intervals) to the database. If you have a table with a time-series column, for example "TimeKey", that's an ideal situation to utilize the tailing approach. Some tables may only contain current information or enrichment data (AKA, no timestamp). In those cases, it's often viable to use a dump of the table. For the purpose of this sample app, when TimeKey was not available, we did a dump of the table.

Navigating through SCCM schema documentation, web content, forums and browsing through SCCM SQL tables – I found valuable data/information. Many valuable use-cases exist.

I decided to take each use-case in logical chunks. For example, I want to gather all data related to services. I defined an DBMON input tail on that table, provided it a sourcetype of sccm-services and capture that table information. One sourcetype per dataset isn't required, but it provided a logical breakdown, in the type of the data I wanted to search for. Example: the data returned from the services table was given a sourcetype of ,sccm-services. For data pulled from the BIOS table, I created a sourcetype of sccm-pcbios.

Lookups

In a structured database world, many databases (SCCM included) may reference to a device in different names/identifiers. For example, your desktop computer might be in a SQL column named "ComputerName" with a value of "DESKTOP". It might also be referenced in the "ComputerID" table, with a value of "2". When you start digging through tables for data, you may find that all of these columns do not exist in each table. You might find ComputerName, ComputerID or both.

As a quick visual example, review the chart layout below.

Table: Users

User	IP Address	Title
honeybadger	192.168.1.50	Ninja

Table: Groups

Group Number	User
1	honeybadger

Table: Group Names

Group Name	Group Number
Administrators	1

If I asked you the question, what's the Group Name of the User Honeybadger, what would you do? You can visually follow how you would get the answer above, but you need to be able to tie this together. Don't worry, we can slice and dice that problem several ways.

The most common approach people take, is to create a SQL Query, full of complex JOIN statements, which can be very resource intensive – while also not being a sustainable long-term solution. This might be the least sustainable route. You're now managing SQL Queries, which may need to be modified over time or creating potential overhead for the JOIN statements on the SQL Server. It's not wrong to take this approach, but just be thinking about what's most suitable.

The second option is to leverage Splunk lookups. Within the App, I've included three lookup generators, which are managed through Saved Searches.

In the scenario above, you will see the App with the following searches.

- Step 1 generates a list of MachineID to ComputerName or Name00, which creations a mapping, since all tables reflect MachineID and not Name.
- Step 2 generates a list of information about users and their attributes and what machines they're responsible for – along with user details
- Step 3 generates a merged relationship list between Step1 and Step2, which allows for you to understand MachineID, ComputerName, UserID and full details about that user.

By following this approach instead, I was able to eliminate a JOIN statement from every single table I'm grabbing data from. More importantly, I was also able to reference |inputlookup in my searches, preventing search overhead on the SH and Indexers, when it made sense.

Props

Structured data is eezy-peezy! You know the format of the data and it's specified in the DBMON inputs, resulting in very little needed from the props.conf.

What's included in this sample props.conf, is the references to lookups by sourcetype. Normally, you might do automatic lookups or db lookups or non-sourcetype lookups or index lookups

Transforms

Could it get any easier than the props file? Yes, it just did. All that's specified in this file is the stanza name (which is the props references) with a map to the actual filename of the lookup file.

Important Saved Searches

We discussed in the lookup section of that I've included some saved search lookup generators. These generate the basic mappings that I highlighted in the lookups section of this post. Please leave these saved searches, or feel free to learn from them and develop an alternative method.

Dashboards

This sample SCCM App comes with several dashboards, which should be very important for contest participants. You might create a new version of an SCCM from scratch or you might take this one and build-

upon it – it's your choice. At a minimum though, we expect any submission to include at least what's in the present version of the sample app today. You can download the sample app and see how these reports were done.

Here's a review of the dashboards and their content:

- SCCM – App Overview
  - Total Events over last 24 hours
  - Timechart of events, by sourcetype over last 24 hours
  - Total count of events, by sourcetype over last 24 hours
- SCCM – Hardware Investigator
  - Form filter by Computer Name
  - User Info: Computer Name, Domain, User, Time Zone, Title, Department, Email Address, Distinguished Name, Time Zone
  - Hardware Info: Manufacturer, Model, Serial Number, BIOS Information, System Type, Number of CPU's
  - Table demonstrating the hardware specifications, hyperthreading status, clock speed, number of processors, etc.
  - Table demonstrating Computer Memory layout: DIMM size, location of RAM on motherboard, total memory
  - Disk Configuration, by drive, filesystem, description, freespace, total disk size and volume serial number/label
  - Visualization of disk space used
- SCCM – Hardware Overview
  - Computers by Manufacturer
  - Most Common Models
  - Mobile vs Laptop
  - Number of Processors
  - Amount of Memory
  - Disk Drives Used
  -
- SCCM – Network Information – Investigator
  - Form filter by ComputerName, UserName or MacAddress
  - Data comes back, based on filter and provides ComputerName, UserName, title, department, NIC manufacturer, adapter type and Mac Address
- SCCM – Network Information – Overview
  - Manufacturer
  - Adapter Types
  - Product Names
  - Service Name
  - Timechart of Total number of MAC Addresses
  - Stats on the most active MacAddress associations
- SCCM – Operating System – Investigator
  - Form filter by ComputerName
  - Basic Information: Computer Name, Operating System, Service Pack, OS Build Number, Installation Date, Last Bootup Time
  - Table identifying the User Locale settings and configuration paths
- SCCM – Operating System – Overview
  - Operating Systems versions
  - Service Pack versions
  - Build Numbers
  - Timechart OS Installation by year
  - Systems not booted in over a "year" or "xyz" timeframe
  - OS Languages
  - Installation and configuration paths, most common and least common
- SCCM – Processes – Investigator

    - TBD – Only 1 report
- SCCM – Services – Investigator
    - Form filter on Computer Name
    - List of Application Name, process name, path name, service type, service mode, start name and service current state
- SCCM – Software – Investigator
    - Form filter by Computer Name
    - Basic Information: Computer, Domain, Username, Time Zone, Last Update
    - Timechart count of installed software over time
    - List of the most recently installed software items
    - All installed software installed on the machine
- SCCM – Software Overview
    - Most common software installed over the last 24 Hours

Tables vs Views

This will be a philosophical debate for each developer working with SCCM. As a general best practice, you should resort to SQL Views for gathering information. They can be clean, are portable between versions and are less taxing than something hitting the data table.

But wait, why is this app designed to hit tables? Reviewing the information through SQL Views, I found that there were some inconsistencies. Specifically, there were the following concerns:

1. Some SQL Views are not populated with all of the columns extracted from the data tables. In several cases, it was a critical column of data
2. Not all SQL Views are populated in SCCM, meaning you would need to setup the SQL Views or find the corresponding root table
3. SQL Views aren't populating all information that is present within the tables, just common ones that people might want to access.

For development speed in this sample app, I mapped to tables, retrieving all the information I needed. In my test instance, against a 30GB populated SCCM database, I never witnessed a SQL query running longer than 4-5 seconds. While this might not be symbolic of everyone's experience, but consideration should be given to where you're pulling the data. (IDEA: Perhaps your inputs have references to both SQL Views and SQL Tables, commented out, allowing the user to decide what's best for them and datasource?)

Missing Extras

As noted before, SCCM is a large software application critical to the Microsoft Enterprise. There are significantly more use-cases than the one described in this document. In addition to all of the information in SQL, there are a significant amount of logs – which contain a variety of information that may/may not be of relevance. Some requests have come up, which are well written in the logs.

Some of these sample logs include the following:

- AppDiscovery.log
- AppEnfore.log
- AppIntentEval.log
- AssetAdvisor.log

- CAS.log
- Ccm32bitLauncher.log
- CcmEval.log
- CcnmExcec.log
- CcmMessaging.log
- CcmNotificationAgent.log
- CcmRepair.log
- CcmRestart.log
- CCMSDKProvider.log
- CcmSqlCE.log
- CCMVDIProvider.log
- CcmSqlCE.log
- CCMVDIProvider.log
- CcmSqlCE.log
- CCMVDIProvider.log
- CertEnrollAgent.log
- CertificateMaintenance.log
- CIAgent.log
- CIDownloader.log
- CIStateStore.log
- CIStore.log
- CITaskMgr.log
- ClientIDManagerStartup.log
- ClientLocation.log
- CmRcService.log
- ContentTransferManager.log
- DataTransferService.log
- DCMAgent.log
- DCMReporting.log
- DcmWMIProvider.log
- EndpointProtectionAgent.log
- Execmgr.log
- ExpressionSolver.log
- ExternalEventAgent.log
- FileBITS.log
- FileSystemFile.log
- FSPStateMessage.log
- InternetProxy.log
- InventoryAgent.log
- InventoryProvider.log
- LocationServices.log
- MaintenanceCoordinator.log
- ManagedProvider.log
- Mtmgr.log
- Oobmgmt.log
- PeerDBAgent.log
- PolicyAgent.log
- PoliceyEvaluator.log
- pwrmgmt.log
- PwrProvider.log
- RebootCoordinator.log
- ScanAgent.log
- Scheduler.log
- ServiceWindowManager.log

- Setuppolicyevaluator.log
- Smscliui.log
- Smssha.log
- SoftwareCatalogUpdateEndpoint.log
- SoftwareCenterSystemTasks.log
- SrcUpdateMgr.log
- StateMessage.log
- StateMessageProvider.log
- StatusAgent.log
- UpdatesDeployment.log
- UpdatesHandler.log
- UpdatesStore.log
- UpdateTrustedSites.log
- UserAffinity.log
- UserAffinityProvider.log
- VirtualApp.log
- Wedmtrace.log
- WUAHandler.log

NOTE

Again, while there might be an important couple of log files, most all of this information resides right in the SQL database – with a deeper level of fidelity. We're not going to suggest one path or another, for gathering and getting the information your app populates with. We just wanted to share the log files that are associated with SCCM, in case one becomes of value.