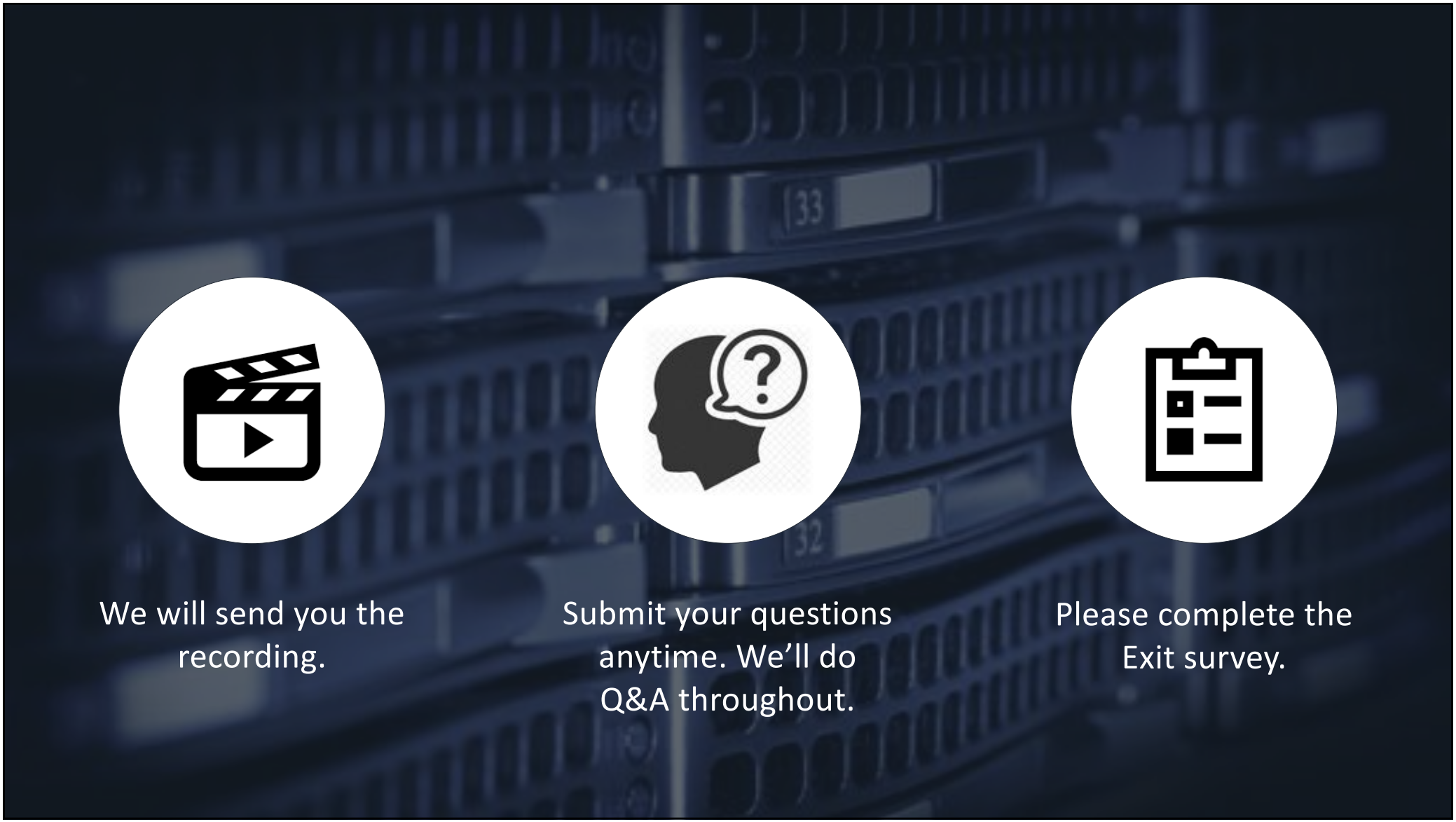
OPMANTEK

NETWORK MANAGEMENT AND IT AUDIT SOFTWARE

Collecting non-SSH/Telnet Device Configurations v1.0 – January 2019

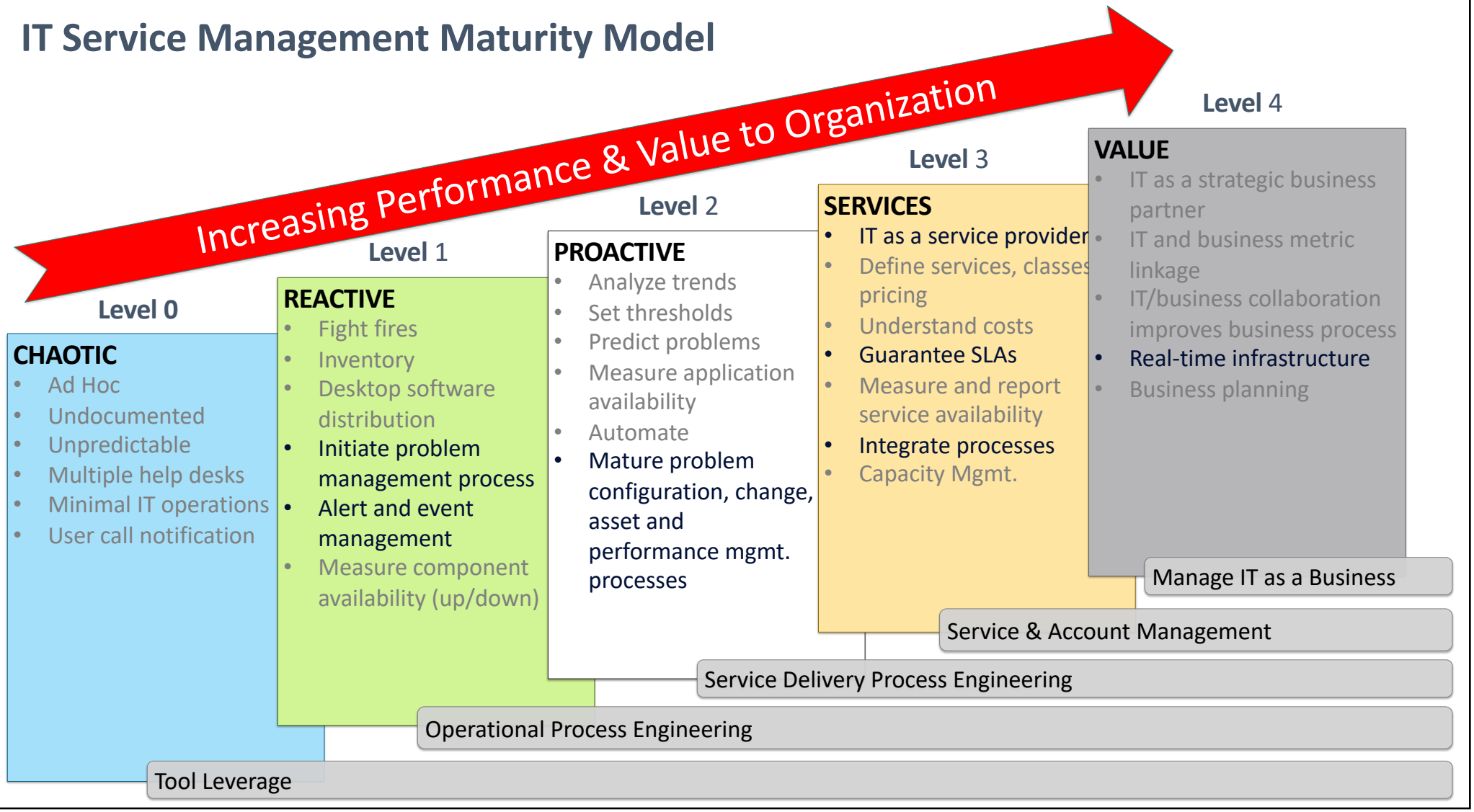

We will send you the recording.
Submit your questions anytime. We'll do Q&A throughout.
Please complete the Exit survey.

Topics for Today

opConfig 3.1.1 introduced the ability to collect or transform configuration data using a plugin architecture. In this webinar we will cover how to use this new feature to collect configuration data from devices that do not have a traditional command-line interface (CLI) accessed through SSH/Telnet. Join us for this in-depth session while we explore –

- How to architect and implement an opConfig Plugin
- Methods for shelling out from the plugin to another program and handle returned data
- Create a simple plugin to read and parse a file (i.e. csv, txt, json, xml, etc) into actionable data
- How to transform configuration data with a plugin
- How to raise and manage NMIS alerts

IT Service Management Maturity Model



Architecting a Solution

Open-Source

NMIS: *Core Performance and Fault Monitoring*

Commerial Solutions

opConfig: *Capture, track and push configuration changes*

Useful References

Where can I go when I have questions?

- NMIS Wiki – https://community.opmantek.com/display/NMIS/Home
- opConfig Wiki – https://community.opmantek.com/display/opconfig/Home
  - Plugins- https://community.opmantek.com/display/opconfig/Plugins+in+opConfig

- Community Questions Board - https://community.opmantek.com/questions
- Support Issues – support@opmantek.com
- Sales – usa@opmantek.com

References

You should bookmark these…

UC Berkley Secure Device Configuration Guideline

https://security.berkeley.edu/secure-device-configuration-guideline

Center for Internet Security

https://www.cisecurity.org/controls/

https://benchmarks.cisecurity.org/en-us/?route=downloads.benchmarks

National Security Agency Security Configuration Guidelines (now hosted at IAD)

https://www.iad.gov/iad/library/ia-guidance/security-configuration/index.cfm

Monitoring for Configuration Changes with opConfig

Configuration Change Detection

Focused Goals

- Configuration changes on covered devices should be logged automatically or via established change management processes
- Resource custodians should be alerted when configuration changes are made on covered devices to allow for identification of malicious activities on covered devices

opConfig can capture, track, push and rollback configuration changes for any network connected device or cloud application.

https://opmantek.com/network-configuration-management-opconfig/

Configuration Change Criteria

When and How are device configurations collected?

- Device Commands
  - Command Collection (which commands are run, and how) are defined in command_sets.nmis
  - How opConfig talks to a device is defined in cli phrasebooks
  - These can (and should) be customized for your environment

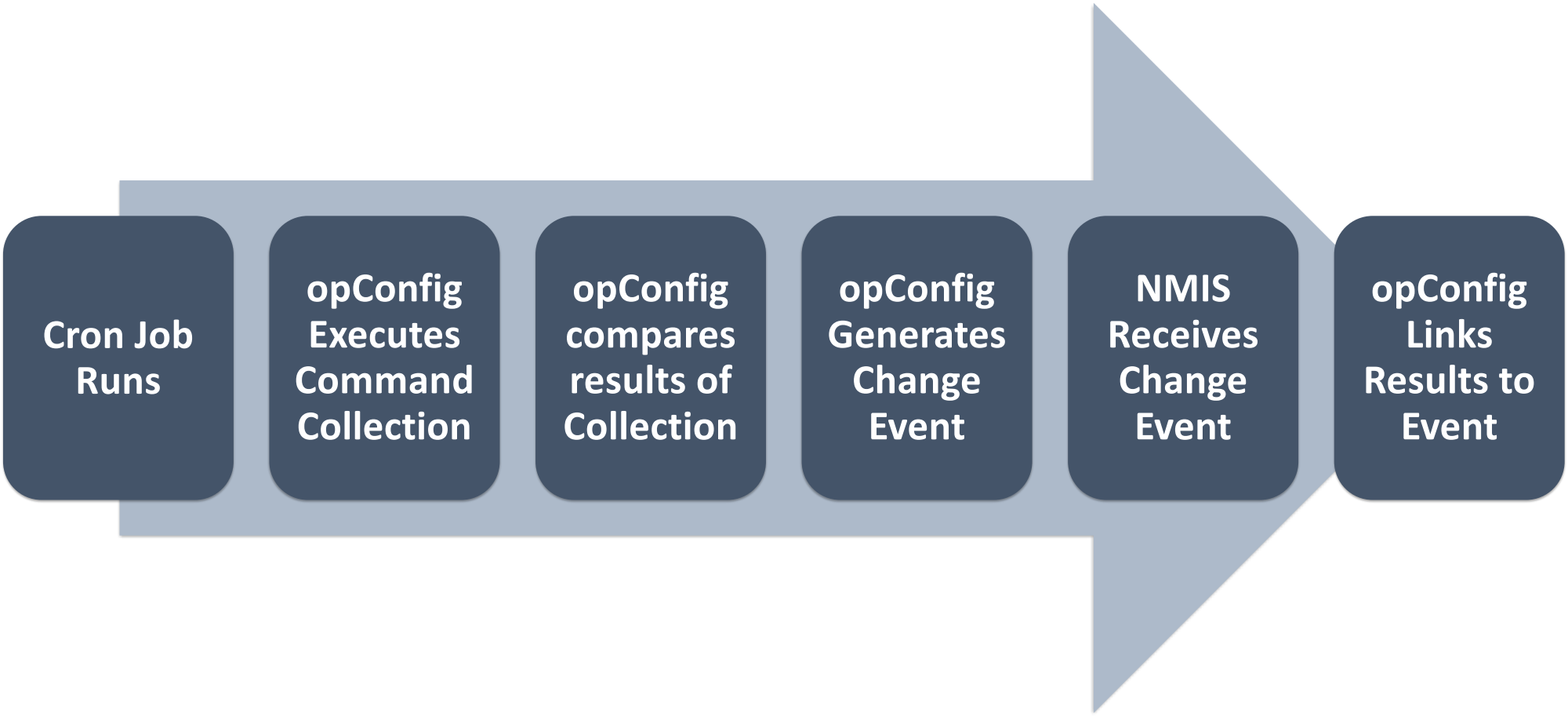
- Scheduled Collection
  - Daily/Hourly scheduled through CRON job
  - Commands are defined as to when they should run (i.e. Daily/Hourly/Troubleshooting)

- On Demand
  - opConfig includes a robust API that can run from the CLI or called via script
  - opEvents can use the opConfig API for command collection for a specified device
  - Can call one command or a group of commands (i.e. Troubleshooting)

https://community.opmantek.com/display/opconfig/Home

opConfig Collection and Alerting

Queued by CRON job (usually command collection)



Common Workflow for Configuration Change Detection

This is a starting point for internal discussion

Introduction to opConfig Plugins

opConfig Plugins Development Patterns

When and Why You Might use opConfig Plugins

- Collecting arbitrary data NOT from SSH/Telnet CLI and storing it
- Looking up and beautifying data
- Running multiple commands and comparing the output to create a result
- Doing weird stuff (technical term) which is only resolved by writing code

opConfig Plugins

Expanding on opConfig's Command Collection System

- Augment and extend the Command Collection functionality
- Two classes
  - Collecting device configurations
  - Processing to filter or transform configuration data collection
- Stored in: /usr/local/omk/conf/config_plugins/

opConfig Plugins

Rules and Restrictions

- Each plugin must be valid Perl (check with perl –cw YourPlugin.pm)
- Plugins must be named X.pm
- Must include a ‘package X;’ line
- Package name must not clash with any opConfig or NMIS components
- Plugin MUST NOT use Exporter to export anything from its namespace
- Plugin MUST NOT use alarm()
- Should include a version declaration right after the package line

Collection

Collecting device configuration data with a plugin

- Create the collection plugin
- Configure one or more Commands to be delegated to that plugin

  ‘command’ => “external command”
  ‘use_collection_plugin’ => “PluginName”
- Only one collection plugin can be assigned per command
- Plugin MUST NOT modify any of the arguments passed to it

Collection

Valid Return Values

- The function must return a hash reference with the following supported keys:
  - success (0 or 1),
  - error (containing an error message that opConfig should handle),
  - ignore (0 or 1),
  - configuration_data (text or other data; i.e. the configuration data that opConfig is to associate with this node and command)

1. if ignore is 1, then the command is ignored altogether and opConfig does not save anything.
2. if success is 1, then and only then the returned configuration_data is processed and stored by opConfig.
3. The error response property is ignored if success is 1, otherwise the error message is logged and reported.

Collection

Sample Collection plugin subroutine calling an external program

```
sub collect_configuration
{
    my (%args) = @_;

    my ($node, $node_info, $command, $credential_set, $logger, $opconfig)
            = @args{qw(node node_info command credential_set logger opconfig)};
    $logger->info("Plugin ".__PACKAGE__." about to collect data for $node, command $command->{command}");
    # maybe we need to shell out to some program?
    open(P, "-|", "/usr/local/bin/complicated_operation", $command->{command}, $node_info->{host})
            or return { error => "failed to start complicated_operation: $!" };
    my $goodies = <P>;
    close P;
    return { error => "complicated_operation failed: $?" } if ($?);
    return { success => 1, configuration_data => $goodies };
}
1;
```

Processing

Filter or transform configuration data

- Create the processing plugin
- Configure one or more Commands calling the Processing plugin

```
'command' => 'df -i',
'use_processing_plugins' => [ "LinuxCommandPlugin" ],
```

- Example 1:
  - Command delegation: /usr/local/omk/conf/command_sets.d/linux.nmis (line 92)
  - Command plugin: /usr/local/omk/conf/config_plugins/LinuxCommandPlugin.pm
- Example 2:
  - Command delegation: /usr/local/omk/conf/command_sets.d/ios.nmis (line 281)
  - Command plugin: /usr/local/omk/conf/config_plugins/HighBandwidthPlugin.pm

Processing

Valid Return Values

- error (error message that opConfig should handle)
- success (0 or 1)
- configuration_data (optional, modified/filtered data that replaces the original input)
- derived_info (optional, a hashref of information derived from the raw input data)
- alerts (optional, hashref of alerts to raise/close),
- conditions (optional, a hashref of condition states)

1. only if success is 1 will configuration_data, derived_info, alerts and conditions be processed by opConfig.
2. If a plugin signals an error, then the error message is logged and reported first, then opConfig continues with other plugins in the pipeline. any other returned data is ignored.
3. If configuration_data is returned, then it replaces the original input and is passed to the next pipeline stage. Otherwise the original input is used.
4. If derived info, alerts or conditions are returned, then they're merged with any already existing information. in other words, each processing plugin can only add to any of these, not overwrite what a previous plugin has reported.

CONTACT FOR FOLLOW UP

Commercial enquiries:

Tom Wiri
Account Executive
+1 (512) 430-4450
usa@opmantek.com

Technical enquiries:

Mark Henry
Senior Engineer
+1 (207) 951-2428
markh@opmantek.com