Snap on Windows

An Intel-sponsored, open-source telemetry framework
IT 447

April 4, 2017

Phillip Anderson

McKade Clements

Devin Durtschi

Mathew Kuhn

Jesse Millar

Coach: Dr. Jay Ekstrom

Sponsor: Taylor Thomas

Table of Contents

Source Code
Constraint Matrix
Governance Framework
Communication
Acceptance Documentation
Gantt Chart

Executive Summary

Snap for Windows is a Brigham Young University Information Technology 2016-2017 capstone project. Snap is an open-source telemetry system headed by Intel, meant to facilitate the remote monitoring of large networks and company infrastructures. Snap works through three types of "plugins," which allow for modularized collecting, processing, and publishing of system metrics. These metrics can include data such as cpu usage, number of processes running on a system, and memory available.

Previously, Snap's functionalities were constrained to Linux systems only, and were not compatible with Windows. This prevented a large portion of company infrastructures from utilizing Snap, as companies typically run the Windows operating system on a large portion of their network. The objective for this project included automating Snap's build process onto Windows through a build script and creating three separate collector plugins based on the Windows' Perfmon, Active Directory, and Sysinternals applications.

The motivation behind this objective was first, to expand Snap's capabilities to allow companies to utilize it on a larger portion of their infrastructure, and second, to reduce required system maintenance/monitoring efforts, thus saving employees' time and saving companies money. Snap is expected to assist in increasing Intel's revenue by making Windows products (which run on Intel processors) more marketable. Although there are similar products on the market, including a native telemetry system built into Windows 10, Snap is unique in that it is extremely modular through the plugin model, it is very customizable through tasks and configuration specifications, and it is highly scalable to large networks.

The customer for the project included Intel, but also the entire Snap community and other companies who will be using Snap. All of their input was included in the plugin and metric choices specified in the project objective. We were able to reach our goal of completing the three plugins mentioned, as well as the build script. The Perfmon and Active Directory plugins have also been merged into the official Snap repository.

Several lessons were learned throughout the course of the project which will be critical for the success of future endeavors. First, open-source projects have a very different nature than start-to-finish projects. Snap involved changing libraries, lack of code ownership, changing documentation, and compliance with Snap best practices. It provided a very real understanding of experiencing success after joining a pre-existing team. Furthermore, Snap involved risks due to the nature of open-source projects. This required that we understand and implement risk-management to the best of our abilities, using a Gantt Chart and really analyzing the risks.

In terms of future work, our plugins will most likely be converted from a Powershell base to WMI. Both are methods of accessing the native Windows system-data API, but WMI has less overhead (although it is more complicated to implement). Other existing Snap plugins will also be ported to Windows, such as Docker. Finally, a machine learning algorithm will most likely be developed so that Snap not only collects, processes, and stores data, but will also be able to *act*

based on that data. The details of each of the key points addressed in this Executive Summary are discussed more extensively below.

Introduction

Telemetry is the gathering of system metrics, or internal critical information, from a distance. This often is used in the business world referring to sales and other business metrics that lead C level executives to pivot the direction in which a company is headed. Alternatively, there are other types of data that can be remotely gathered to help in decision making, such as in the auditing of a company's technical infrastructure. Having data on the computing assets of a business is useful for the purposes of preemptive action or immediate reaction to scenarios that may cause harm or complications to the function of the system in question. For example, measuring the number of established TCP connections there are on individual machines forming a cluster of file servers makes it possible to analyze and then make decisions for load balancing or troubleshooting root causes of debilitating issues. This allows the servers to continue uninterrupted in their designated tasks and maximizes their productivity. All businesses are looking for better and cheaper ways to use telemetry to make these types of daily decisions.There are not many solutions currently on the market to fill this need. Those that do exist are costly or are complicated enough to discourage a wide range of users from adopting them. Ad-hoc methods are then implemented and using cron jobs and scripts or Powershell, and often crucial metrics can be lost.

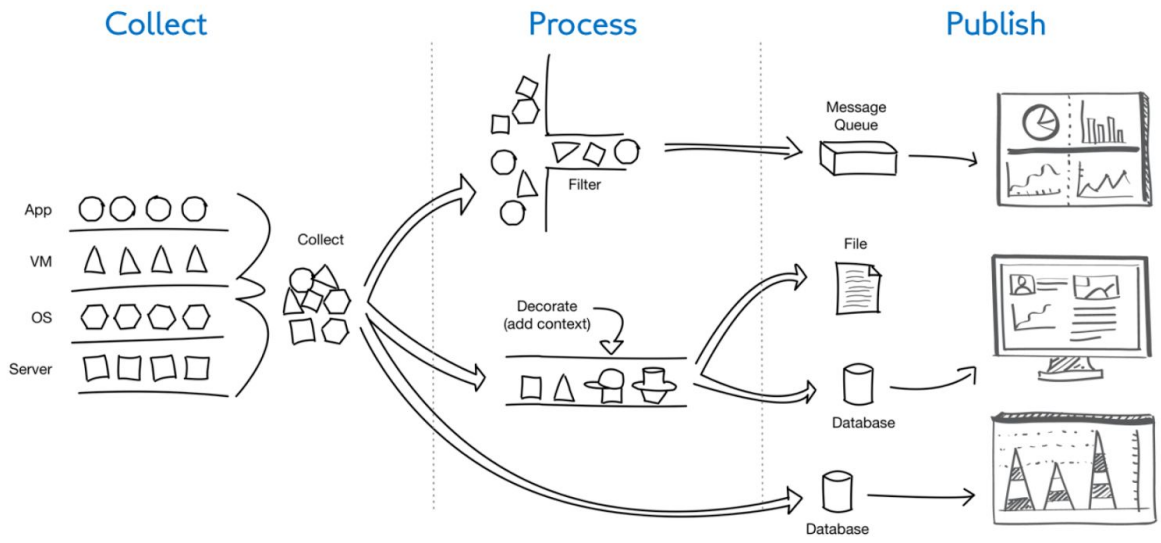
Snap is an open source project started in July of 2015 with the intent of creating a single API for collecting telemetry data in Unix-based systems. Snap also allows for collected data to be processed as well as published. Processing data involves adding context, or "data about data," to the collected information. It also allows for generating new information from the data collected, such as a calculated average of a certain metric across all devices in a cluster. This can then be published, or presented in a logical and directed way, and can be analyzed by the user in the form of text files, database outputs, or analytic platforms. By fully conglomerating this process of collecting, processing, and publishing telemetrics Snap simplifies the utilization of this internal data and gives this power to anyone that needs it.

The problem that Snap has come up against in its development has been the limiting factor of being Unix unique while many businesses utilize Windows based systems. That is why Intel came to us to get some help in expanding the base of this already versatile software. Snap needed a jumpstart in breaching the platform barrier, and that is what we have done. This document was created as a part of the Brigham Young University Information Technology capstone class of Fall 2016-Winter 2017. It outlines the process that was used to port the Snap Open Telemetry framework to the Windows Platform and the finished products being several plugins for the collection of Windows specific application's metrics.

Concept Definition

Background

Plugins are how Snap collects and manipulates system data. Plugins are modular programs, written in Go, which can be mixed and matched to allow for customized metric output. Plugins are grouped into collector, processor, and publisher categories and run on top of the main Snap process. The data from the plugins is accessed through the Snap API. Plugins can be built and ran as needed to collect whatever type of telemetry data system administrators may need, making them a critical component to Snap.



Plugins are run using “tasks.” Tasks define the data collector, the amount of time the collector should run (whether infinitely or for a certain duration), the plugin used to process the data, and the publisher plugin which outputs the data in a certain format. Tasks can also define the metrics gathered from a plugin. For example, the netstat plugin may include metrics for open and established TCP connections but the task may only require the collecting of open TCP connection information.

There are a number of products similar to Snap. Nagios, which performs monitoring of network devices, is one example of a Snap alternative. Snap’s strength lies in how modular and streamlined it is. Windows has its own telemetry collection tool, but it is limited to Windows-based networks and is thus not as scalable as Snap.

Because the Snap project is open source, the code behind Snap is available in a GitHub repository where a thriving community actively works to make Snap better. Common contributions are bug fixes, additional plugins, and sharing of information through searchable discussions. The repository is well documented and contains deployable Snap binaries.

Prior to this project, Snap plugins were only built for the Unix environment. However, there has been a large push by the Snap community, which not only includes individuals but also large corporations like Intel and Staples Inc., to make it possible to use Snap on Windows-based systems. Since Snap is written in the Go programming language, some of the existing Snap plugins, like netstat, could have cross-compiled for Windows. However, in order to make it worthwhile to users, a full-featured Windows port was necessary. This meant plugins needed to be written to gather telemetry data from Windows programs like Active Directory and Perfmon.

Our task, as defined by our Objective Statement, was to automate the Snap installation process and port at least three Snap plugins to Windows. At the completion of our project, the install process is automated and painless and we have built Perfmon, Sysinternals, and Active Directory plugins which have been accepted into the official Snap repository.

Stakeholders

The following table is a summary of each of the stakeholders, including the role of each party within Snap, and the benefits received by each party upon completion of our project. Below the table are brief, high-level descriptions of each stakeholder.

Stakeholder	**Interest**	**Benefit**
Intel	Creators of Snap and sponsor of project	Increased sales; double market share by porting to Windows
Companies with Windows Systems	Building and using plugins for their own monitoring needs	Easier monitoring and metric-collecting; Less need to hire for monitoring
Operators of Windows Systems	Contributing to Snap plugins on behalf of companies	Makes managing systems easier
Open-Source Maintainers of Snap Project	Participating in Snap creation and maintaining	Economic benefits include selling more Intel products which run Windows and monitoring environments more easily
Members of the Snap Community	Participating in Snap community; May be members of above stakeholder groups as well	Learn more about telemetry

Intel

Intel holds a direct business interest in getting Snap to work on Windows because Windows is run almost exclusively on Intel processors. Having more functionality added to Windows helps

Intel sell more of their own products. By improving upon a telemetry framework specifically designed to monitor Intel devices, customers can use Snap to reduce the labor needed to monitor systems, thus reducing their total cost of ownership.

Companies with Windows Systems

Companies with existing Windows systems could benefit greatly from a more concise tool for telemetry. Utilizing Snap allows companies more flexibility and adaptability by making it easier to integrate with Unix based systems should they desire to "mix and match" or eventually migrate from Windows systems to Unix systems.

Native Windows support in Snap reduces the total cost of ownership for third-party companies by limiting the need to train operators in more than one telemetry system or having several operators to maintain separate systems.build process as well

Operators of Windows Systems

Distinct groups of operators such as app developers, system administrators, lab managers, and database administrators can all use Snap to monitor system performance of the products they build and administrate.

Open-source Maintainers of the Snap Project

Maintainers of the open-source Snap project are the main shareholders. The maintainers are the ones that control which plugins and features are added to the Snap code base. Project maintainers have an interest in seeing that the project functions properly as a whole.

Members of the Snap Community

Members of the Snap community use Snap and are not actively involved in the development of the project. Their interests revolve around the ability to more easily and quickly gather, process, and publish data from their systems in a user-friendly format. Regardless, these users have a vested interest in keeping the project going because Snap is a part of their systems.

Stakeholder Requirements

The following is a list of requirements as obtained from each of our stakeholders and how we fulfilled each requirement:

- **Snap executable compiled for Windows**
    - A Snap executable was requested so that pre-built binaries can be distributed to those who would like to use Snap, rather than mandating that users figure out the compile process. To fulfill this requirement, we worked with Snap maintainers to modified the existing Snap installation scripts to work on Windows. We do not have access to the Snap build pipeline, but Snap project maintainers can use our

scripts to automatically generate Snap executables for distribution on the official website.

- **Automated build script to compile Snap for Windows**
    - The process to compile Snap for Windows should be automated according to stakeholder requirements. This will allow for members of the Snap community to more easily compile Snap, rather than going through a more tedious and complicated manual process. As mentioned above, we worked with Snap maintainers to modify existing Linux build scripts to function on Windows. These scripts have been accepted into the official repositories.
- **Plugins written in Go**
    - The Windows-specific plugins we were asked to write were written in Go to provide greater maintainability for the project maintainers, as they are most familiar with this language.
- **Plugin functionality written in language native to Windows**
    - The gathering of system metrics on Windows was requested to be done via Windows-default methods. This left us with the options of either PowerShell or Batch scripting, as these languages provide native interfaces into the Windows API's and allow for greater ease in gathering the metric data. We largely utilized PowerShell and created plugins that met stakeholder requirements.
- **Follow plugin-authoring framework**
    - To simplify the process of merging our code into the official Snap repositories, we were asked to observe and respect the interfaces of the plugin libraries. This was slightly more complicated than anticipated because of the deprecating of the initial framework halfway through the project, but we were able to meet stakeholder demands by quickly pivoting to the new framework.
- **Sysinternals plugin**
    - The Sysinternals plugin, as requested by the Snap community, provides metrics associated with common Sysinternals tools. It integrates seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.
- **Perfmon plugin**
    - Perfmon was specifically requested by a member of the Snap community and, per the request, provides common metrics associated with the native Windows Perfmon graphical application, such as CPU utilization, memory available, and memory used. It integrates seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.
- **Active Directory plugin**
    - The Active Directory plugin was also requested by interested parties and provides desired metrics from Active Directory. It should integrate seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.

- **Testing of each plugin**
    - The plugins were tested on our local machines and have unit tests written to verify that the functionality remains stable from release to release. This simplifies the maintenance and deployment process for Snap maintainers.
- **Issue logging system**
    - To satisfy the class requirement of issue tracking, our team utilized the GitHub issue logging system throughout the project. This made our team interaction more fluid and will allow the project to be more maintainable after we graduate.
- **Project documentation**
    - To satisfy further class and stakeholder requirements, any special bits of information needed to work with our plugins have been recorded and shared for maintainability purposes. Existing documentation has been modified as well, to include instructions for using Snap on Windows platforms in addition to Unix-based platforms.

Validation

Throughout the early stages of our project, we spoke mostly with our sponsor, Taylor Thomas, and discussed each of the above requirements during the course of several video-conferences. As we gained a better understanding of the requirements and presented them to Taylor during these meetings, he was able to approve each of them. At the conclusion of our project, Taylor has approved personally all our code and methods and worked with his superiors at Intel to obtain final approval.

Verification

Now that our code is complete, we have met and conversed with several Intel representatives to review the requirements and request feedback in terms of what we may need to modify. Slight modifications were made and documented on the GitHub repository and formal approval was given. The code we've written is now merged into the main Snap repositories.

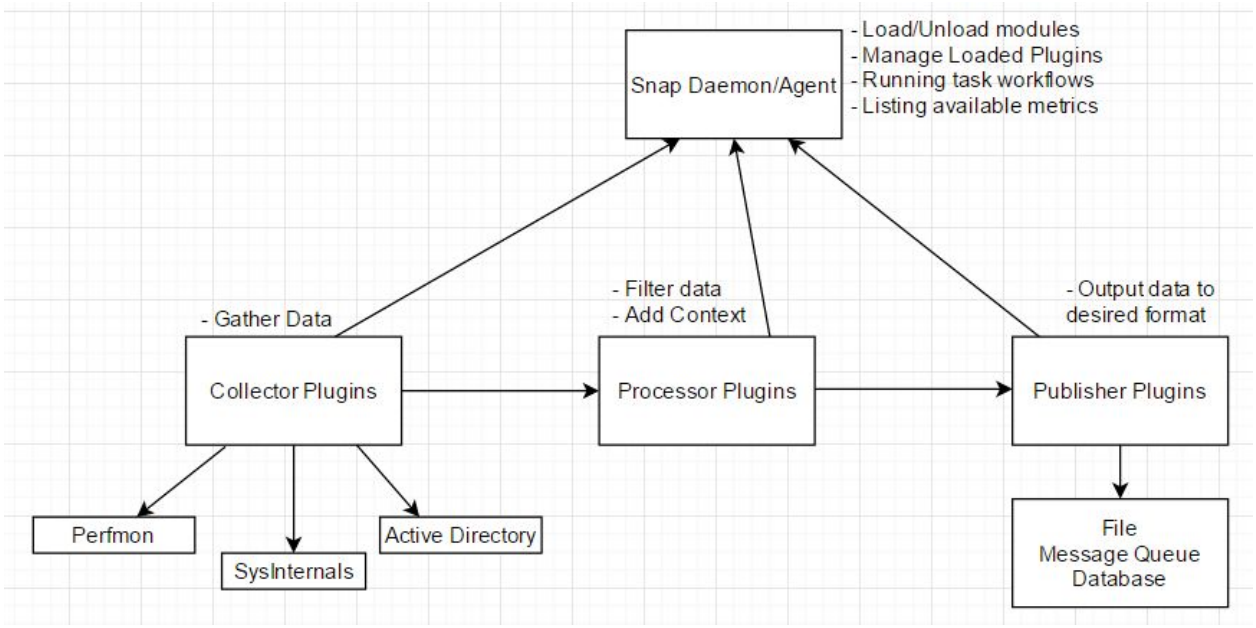
System Definition

System Requirements

- **Snap executable compiled for Windows**
    - The Windows executable should have the same functionalities as the binary build for Unix systems
- **Automated build process to install Snap on Windows**

    - The installation process should be as simple and standard as possible to allow as many system administrators as possible to install Snap quickly
- **Plugins written in Go**
    - The code for our plugins needs to be readable and easily maintainable
    - Our code should be simple to cross-compile
- **Plugin functionality written in language native to Windows**
    - Our code should work well in Windows environments
    - Metrics should be gathered and manipulated in the most efficient way possible to avoid introducing latency into client systems
- **Follow plugin-authoring framework**
    - Plugins must integrate well with existing Snap framework
    - Plugins should be completely modular
- **Sysinternals plugin**
    - Must pass PsUtils data to Snap
- **Perfmon plugin**
    - Must gather CPU and memory utilization metrics
- **Active Directory plugin**
    - Build and maintain a documented development environment which implements Active Directory
    - Retrieve, parse, and send Active Directory usage data to Snap
- **Testing of each plugin**
    - Write small unit tests as mandated by the
    - Development environment which implements a domain controller
    - Development environment implements Active Directory
- **GitHub issue log**
    - When we finish the capstone project, we should leave behind enough documentation and searchable discussion about our project, code and the issues we hit as possible
- **Project documentation**
    - Documentation is stored in public and searchable GitHub wiki format

Logical Architecture



Project Component Details

In order to fulfill our objective of porting Snap from Unix-based environments to Windows in a way which is maintainable by the project maintainers, we completed the following tasks. The progress we made for each major aspect of our project is technically detailed below:

Build Windows Test Environment

Because we are working within a university setting, the current systems to which we have access are all production environments that have important functions to the school. To be able to test our developments to Snap on the Windows platform we needed a separate environment that was isolated from the school's network to protect these production environments from possible damage of services from an untrusted domain within the same network.

To do this we have built a virtualize testing environment on a server using VMWare ESXi 6 which implements a domain controller as well as Active Directory in order to fully test each plugin in a cluster setting. This also allowed us to test the Active Directory plugin. To isolate this domain from the main IT department domain we have virtualized a separate vlan within the ESXi that works as an internal LAN that connects out to the external WAN through a pfSense firewall configured to allow Remote Desktop through to each of the nodes within. There are three Windows 10 nodes and a Ubuntu 16.04 Server node that are all joined to the Active Directory domain controller.

Perfmon Plugin

The Perfmon plugin is based on the Windows Perfmon application, which provides a graphical interface to Windows systems metrics. These metrics include items such as cpu-usage, memory available, memory committed, etc. This plugin collects 12 of these metrics through the Snap Framework.

The plugin is based around GoLang (in order to conform with Intel best practices) and Windows PowerShell. Version 3.0 or higher of PowerShell is needed as the script uses the "Get-Counter" cmdlet. This particular cmdlet enables the plugin to access Windows native API, which provides access to the perfmon metrics. As PowerShell inherently comes with large amounts of overhead, GoRoutines were utilized to add concurrency to the plugin. GoRoutines act as very lightweight threads to increase execution speeds. Parallelism was also added to increase speeds, by executing the plugin in separate, parallel instances, based on how many processors the system has. This brought the time is took to retrieve all 12 plugins down from 23 seconds to 3 seconds. As the added concurrency would then occasionally lead to intermittent failures, a mutex was added to the base metric-storing map used in the code. The mutex locked the map until a particular thread was finished executing, and then released the map for another thread to use.

The rest of the Perfmon plugin code followed Snap best practices in terms of integrating the functionality with Snap. Unit tests were created to ensure correct functionality of the plugin, and integration tests were performed by simply running the plugin through Snap. After extensive code review and revisioning, the Perfmon plugin has been merged with the official Intel repository.

Sysinternals Plugin

The Sysinternals plugin uses an executable called PsList. This file comes from Microsoft's Sysinternals suite which has about 40 executables relating to system management. PsList is used to give detailed information about processes currently running on the Windows machine. This plugin is utilized to gather information about the total number of processes, total number of threads, and total number of handles. The plugin initially checked for the PsList executable and downloaded it from the Microsoft website if necessary. The community developers found this feature to be too intrusive and felt that those using the plugin should download it themselves.

This plugin has its own unit tests and is has been tested to ensure that it functions with Snap properly. Developers in the Snap community have looked at the plugin and submitted feedback for it.

Active Directory Plugin

The Active Directory plugin followed a very similar path to completion as the Perfmon plugin. The only difference is that we were given specific metrics to gather for this plugin by the Snap community. Upon completion, the Active Directory plugin collects 19

different metrics, including metrics such as DRA (Directory and Resource Administrator) Outbound Bytes, Kerberos Authentications, and and LDAP Client Sessions.
The Active Directory plugin followed a very similar path to completion as the Perfmon plugin. The only difference is that we were given specific metrics to gather for this plugin by the Snap community. Upon completion, the Active Directory plugin collects 19 different metrics, including metrics such as DRA (Directory and Resource Administrator) Outbound Bytes, Kerberos Authentications, and and LDAP Client Sessions.

Like the Perfmon plugin, the Active Directory plugin utilizes GoLang, PowerShell, concurrency, and parallelism. It includes unit tests and has been tested with Snap itself. Due to overhead restraints, we have recommended gathering only 16 metrics at a time, but in future revisions (where WMI will be used in place of PowerShell), all metrics should be collectable at once. The plugin has been code reviewed, revised, and merged into the official Snap repository.

Create an Automated Build Script for Snap

Existing scripts have been modified to automate the build process so that future users who wish to compile their own executables do not have to follow a complex manual process, thus minimizing risk. These changes to the build scripts have been merged with the official Snap repositories.

Critical Path

The critical path for our project began with gathering a basic understanding of the existing Snap framework and getting introduced to the community. From there, we started to manually compile Snap for Windows, as that was required for anything else to work. Once we understood that process and had Snap running on Windows machines, we divided into sub-teams. One sub-team focused on building an automation script to compile Snap automatically. Two other teams focused on writing the three plugins. The final team setup a testing server environment with necessary components, perform plugin testing, and document necessary data relevant to maintaining our plugins. As each sub-team finished their initial task, they would assist other teams with remaining tasks. At the completion of each major task, we created a pull request on GitHub to merge our code with the main Snap repository, which officially signified that task was complete.

Verification and Validation

In order to verify and validate the fulfillment of each system requirement, we submitted our completed automated build process for Windows, as well as our completed plug-ins, to the Snap community via Pull requests on GitHub for approval. This included documentation and unit tests for each feature. In the case that adjustments were needed, the Snap community notified us via the GitHub pull request and we made the necessary adjustments. Upon approval, the community

merged our contribution into the existing Snap repository signifying it was up to the community's standard.

Project Management

Objective Statement

Automate the Snap build process and port at least three Snap data collection plugins for Windows, including Perfmon, Sysinternals, and Active Directory by March 20th.

List of Deliverables

Our project of porting Snap and a collection of plugins to Windows has been completed and the Snap project maintainers have accepted our code for each of the plugins and merged them into the master branch of their GitHub repository. As part of our pull requests that presented our work to the project maintainers, we provided the deliverables listed below.

- Plugin documentation describing the functionality of our newly-built plugins
- Unit tests for our new plugins to automatically verify that the plugins work as intended
- A script to automatically build Snap for a windows machine
- A collection of Snap plugins built for Windows systems

In order to complete the academic aspects of our capstone project, we provided the deliverables below.

- Project presentation slides for our final presentation
- A project video outlining our project in a concise and engaging way

Conclusion

Through Snap's improved way of collecting metrics through a single API, adding this functionality to Windows systems greatly improves troubleshooting and monitoring on those systems. Even someone with minimal system administration skills can now install Snap to Windows systems through an MSI and begin collecting crucial data with the plugins that have been created through this project. Those that are more technical can take this base we have provided and build upon it to gain functionality that will further Snap's capability to perform even more fully in the Windows' world.

Possible future work includes porting more plugins from Linux to Windows (ie. Docker). As our plugins use a PowerShell base, and PowerShell involves substantial overhead, the maintainers expect to convert the PowerShell code to a WMI implementation. WMI is more complex, but requires less overhead than PowerShell. Furthermore, a machine learning algorithm will most

likely be created and implemented. This algorithm will enable Snap to make certain decisions based on the gathered data, further facilitating the monitoring of large infrastructures.

Working in the open source realm is something different that the other projects will not deal with, but that we found challenging and satisfying. There is a wide horizon in front of Snap and because it is an established open source project with an actively involved community, there is much that could happen with it now that we have bridged a significant gap.

References

[1] Cheney, D. (n.d.). Cross compilation with Go 1.5. Retrieved October 05, 2016, from http://dave.cheney.net/2015/08/22/cross-compilation-with-go-1-5

[2] Command go. (n.d.). Retrieved October 05, 2016, from https://golang.org/cmd/go/

[3] Develop Perfmon windows performance monitor plugin · Issue #1175 · intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from https://github.com/intelsdi-x/snap/issues/1175

[4] Go by Example. (n.d.). Retrieved October 05, 2016, from https://gobyexample.com/

[5] How to Write Go Code. (n.d.). Retrieved October 05, 2016, from https://golang.org/doc/code.html

[6] Intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from https://github.com/intelsdi-x/snap/blob/master/docs/BUILD_AND_TEST.md

[7] PLUGIN CATALOG. (n.d.). Retrieved October 05, 2016, from https://github.com/intelsdi-x/snap/blob/master/docs/PLUGIN_CATALOG.md

[8] SNAP REST API. (n.d.). Retrieved October 05, 2016, from https://github.com/intelsdi-x/snap/blob/master/docs/REST_API.md

[9] Support flow collector like sflow or netflow? · Issue #1030 · intelsdi-x/snap. (n.d.). Retrieved October 05, 2016, from https://github.com/intelsdi-x/snap/issues/1030

[10] Windows support · Issue #671 · intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from https://github.com/intelsdi-x/snap/issues/671

Appendix

Source Code

Note: As the code bases are very large for each plugin, we are simply providing links to their GitHub repositories as instructed. The official repository versions are Private right now (they

are not viewable unless you have been invited to view the repo), so these will be placed in parenthesis (if available), with secondary repository for each listed below:

Perfmon plugin: https://github.com/Snap-for-Windows/snap-plugin-collector-perfmon (https://github.com/intelsdi-x/snap-plugin-collector-perfmon)
Sysinternals plugin: https://github.com/Snap-for-Windows/snap-plugin-collector-sysinternals
Active Directory plugin: https://github.com/Snap-for-Windows/snap-plugin-collector-active-directory (https://github.com/intelsdi-x/snap-plugin-collector-active-directory)
Build scripts: https://github.com/intelsdi-x/snap/blob/master/scripts/build_all.sh

Constraint Matrix

	Scope	Schedule	Resources
Most Constrained		**X**	
Moderately Constrained	**X**		
Least Constrained			**X**

The schedule of our project is by far its most constrained aspect in our constraint matrix. We simply must have the project completed by mid-March in order to be prepared to present our work at the end of the capstone class. The scope of the project is moderately constrained because we are making a port of existing code, not adding sweeping new features.

Our resources are nearly limitless within the scope of the project. We have access to an active communication medium with the creators and maintainers of the project, a direct line of communication to our sponsor, full-featured Snap documentation, and, hopefully, far more time than we will need to complete our additions.

Governance Framework

In order for the project to progress as smoothly as possible, we have agreed upon a governance framework. Utilizing this governance framework to decide on individual roles inside the team now will make enforcing team responsibilities easier than if conflicts were resolved when they arise later.

Any and all conflicts among team members or pertaining to project decisions will be resolved by team majority vote. These decisions become final when voted upon and written down in the GitHub project manager. If the conflict involves the open-source code we are contributing to, the issue should be resolved by leaders of the open-source project via Slack or discussion in a GitHub issue thread. As contributors to the project, we want to ensure that we are following the procedures already established by the Snap community and not stepping outside the bounds provided to us.

We will meet as a team twice a week, once with only the students on the team, and the other with the students and coach. The first meeting will happen on Wednesday evenings and the second will happen on Friday mornings. The Wednesday meeting must be attended by all members of the team either in person or via Google Hangouts. The Friday meeting should be attended in person by all team members. We will meet with our sponsor bi-weekly via the Zoom team collaboration application. As member availability may vary for these Zoom meetings, only members who are able to attend will be required to participate. If members consistently miss any of these meetings, we will speak as a group and decide upon a time that works better for everyone.

Communication

Frequent, clear, and searchable communication is integral to the success of this project. For questions regarding the needs of the Snap community or general Snap discussion, we will use Slack. Slack is the main form of communication currently used by the project maintainers at Intel. Google Hangouts will be used for capstone team communication, remote meetings, and general discussions between team members. As an alternative means of communication, a Slack channel has been created specifically for our team's use.

Our meeting notes, personal notes, and other class-related documentation will be kept in a shared Google Drive folder. All other documentation will be housed and maintained on our GitHub project's wiki. GitHub's project management features will be used to keep track of our task list and critical path. GitHub provides a number of tools that make tracking the progress of fixing bugs and adding features very fluid.

Acceptance Documentation

As part of the contribution guidelines, the Snap project maintainers request a single GitHub pull request with one commit per feature added. To accomplish this, we have created a GitHub organization that includes every member of our capstone team and forked the Snap project into our organization. Forking a project in GitHub makes a copy of the code as it currently stands and enables us to modify files without restriction.

When we are done adding and testing a specific feature, we will use GitHub's interface to create a pull request. Creating a pull request sends a notification to the Snap project maintainers informing them that there is new code for them to review and potentially merge into the master branch of their project. Once our code is accepted and merged, our changes will be made available to the community in the precompiled Snap binaries.

Additionally, we will need to draft documents outlining our feature addition proposals to give to our sponsor. This step can be seen as a preliminary requirement approving of our plans before we set off in a potentially wrong direction.

Gantt Chart

Unique ID	Phase	Activity	Task Name	Duration	Start	Finish	Predecessors
1	Planning						
2			Read Snap Documentation	1 wk	05-09-2016	09-09-2016	
3			Write initial project document	1 wk	12-09-2016	16-09-2016	
4			Complete Final Document/define requirements	3 wks			
5	Team						
6			Learn Go	2 wks	12-09-2016	23-09-2016	
7			Learn Github Project Management	2 wks	12-09-2016	23-09-2016	
8			Analyze existing Snap plugins	3 wks	19-09-2016	07-10-2016	
9			Install Windows locally for development	3 wks	26-09-2016	14-10-2016	7,6
10			Understand Glide package management	1 wk			
11	Prototype						
12			Find PowerShell script for Perfmon	1 wk	03-10-2016	07-10-2016	
13			Find PowerShell script for Sysinternals	1 wk	03-10-2016	07-10-2016	
14			Get images for slideshow presentation	1 wk	31-10-2016	04-11-2016	
15	Implementation						
16		Build Test Environment					
17			Install ESX	3 wks	26-09-2016	14-10-2016	
18			Install firewall	4 wks	17-10-2016	11-11-2016	
19			build internal domain	3 wks	14-11-2016	02-12-2016	
20			Get licensing for servers and nodes	3 wks	28-11-2016	16-12-2016	
21			Elevate role of server to Domain Controller	1 wk	19-12-2016	23-12-2016	20
22			Add roles for AD and others mentioned by community	2 wks	26-12-2016	06-01-2017	21
23		Build Script for Windows					
24			Research Makefiles	1 wk	19-12-2016	23-12-2016	
25			Investigate CMake	1 wk	19-12-2016	23-12-2016	

26			Investicate Batsh	1 wk	09-01-2017	13-01-2017	
27			Build dep.bat	6 wks	16-01-2017	24-02-2017	26
28			Build build_snap.bat	6 wks	16-01-2017	24-02-2017	
29			Build build_plugin.bat	6 wks	16-01-2017	24-02-2017	
30			Build build_plugins.bat	6 wks	16-01-2017	24-02-2017	
31			Build build_all.bat	5.2 wks	23-01-2017	27-02-2017	
32		Sysinternals Plugin					
33			Research different sysinternals tools	1 wk	07-11-2016	11-11-2016	
34			Research different ways to install sysinternals tools	1 wk	07-11-2016	11-11-2016	
35			Set up development environment	1 wk	14-11-2016	18-11-2016	
36			Run tool from Go	1 wk	14-11-2016	18-11-2016	
37			Collect Output from Go	1 wk	21-11-2016	25-11-2016	
38			Create schema for parsing from tool	1 wk	05-12-2016	09-12-2016	
39			Parse Output from tool	4 wks	19-12-2016	13-01-2017	
40			Pass data into Snap functions	2 wks	30-01-2017	10-02-2017	
71			Deploy in test dev env.	2 wks	13-02-2017	24-02-2017	40,39,38,37,36,35
41			Create documentation for plugin	1 wk	06-02-2017	10-02-2017	
42			Get plugin accepted by maintainers	5 wks	26-02-2017	30-03-2017	42,41
43			Implement changes requested by maintainers	5 wks	24-02-2017	30-03-2017	
44		Perfmon Plugin					
45			Understand data flow of rand plugin	4 wks	07-11-2016	02-12-2016	
46			Run a test plugin with new plugin lib	4 wks	09-01-2017	03-02-2017	
47			Create dummy perfmon plugin	2 wks	23-01-2017	03-02-2017	
48			Get dummy plugin to run with Snap	2 wks	23-01-2017	03-02-2017	
49			Create PowerShell script to get 3 counters	2 wks	09-01-2017	20-01-2017	

50			Create Go package to get PowerShell data	2 wks	16-01-2017	27-01-2017	
51			Integrate with Glide	2 wks	23-01-2017	03-02-2017	
52			Create unit tests for single system	1 wk	06-02-2017	10-02-2017	52
53			Deploy in test dev env.	1 wk	13-02-2017	17-02-2017	53,52,51,49,48,47,46
54			Create documentation and comment code	5 wks	23-01-2017	24-02-2017	
55			Get plugin accepted by maintainers	7 wks	20-02-2017	07-04-2017	54
56			Implement changes requested by maintainers	7 wks	13-02-2017	31-03-2017	
57		Active Directory Plugin					
58			Research Metrics to be collected with AD	1 wk	16-01-2017	20-01-2017	
59			Find and write Powershell scripts	2 wks	16-01-2017	27-01-2017	
60			Build Go script to run powershells	2 wks	30-01-2017	10-02-2017	60
61			Integrate script with Snap	2 wks	06-02-2017	17-02-2017	
62			Run tests in test environment with functioning AD	2 wks	13-02-2017	24-02-2017	61
63			Write documentation and comment code	6 wks	17-01-2017	27-02-2017	
64			Get plugin accepted by maintainers	5 wks	28-02-2017	03-04-2017	64,63
65			Implement changes requested by maintainers	5 wks	13-03-2017	14-04-2017	
66		Windows Docker Plugin					
67							
68	Closure						
69			Documentation of Work Done	3 wks	13-03-2017	31-03-2017	
70			Final Pull Request	4 wks	03-04-2017	28-04-2017	70