SysFlow Telemetry Pipeline

Release 0.1

Jul 30, 2020

Contents:

CHAPTER 1

Introduction

The SysFlow Telemetry Pipeline is a framework for monitoring cloud workloads and for creating performance and security analytics. The goal of this project is to build all the plumbing required for system telemetry so that users can focus on writing and sharing analytics on a scalable, common open-source platform. The backbone of the telemetry pipeline is a new data format called SysFlow, which lifts raw system event information into an abstraction that describes process behaviors, and their relationships with containers, files, and network. This object-relational format is highly compact, yet it provides broad visibility into container clouds. We have also built several APIs that allow users to process SysFlow with their favorite toolkits.

The pipeline can currently be deployed using docker or kubernetes through helm charts (see sf-deployments project). We plan to support other deployment options, such as OpenShift in the future. Contributions are welcome!

Lastly, C++ and Python APIs are available in the sf-apis project, allowing users to interact with SysFlow traces programmatically. There are also Apache Avro schema files for SysFlow so that users can generate APIs for other languages, such as golang or JAVA.

To learn more about each project, please check the table of contents below, or visit the READMEs in each project's git repo.

This an ongoing research project. We welcome feedback, bug reports, and contributions!

CHAPTER 2

Keep in touch

Please connect with us on our Slack community! For bugs and feature requests, please check our issue tracker.

CHAPTER 3

License

SysFlow and all projects are released under the Apache v2.0 license.

3.1 Quick Start

We encourage you to check the documentation first, but here are a few tips for a quick start.

3.1.1 Starting the collection probe

The easiest way to run the SysFlow collector is from a Docker container, with host mount for the output trace files. The following command shows how to run sf-collector with trace files exported to `/mnt/data` on the host.

```
docker run -d --privileged --name sf-collector \
         -v /var/run/docker.sock:/host/var/run/docker.sock \
         -v /dev:/host/dev -v /proc:/host/proc:ro \
         -v /boot:/host/boot:ro -v /lib/modules:/host/lib/modules:ro \
             -v /usr:/host/usr:ro -v /mnt/data:/mnt/data \
             -e INTERVAL=60 \
             -e EXPORTER_ID=${HOSTNAME} \
             -e OUTPUT=/mnt/data/    \
             -e FILTER="container.name!=sf-collector and container.name!=sf-exporter"␣
↪\
             --rm sysflowtelemetry/sf-collector
```

where INTERVAL denotes the time in seconds before a new trace file is generated, EXPORTER_ID sets the exporter name, OUTPUT is the directory in which trace files are written, and FILTER is the filter expression used to filter collected events. Note: append `container.type!=host` to FILTER expression to filter host events.

Instructions for `docker compose` and `helm` deployments are available in here.

3.1.2 Inspecting collected traces

A command line utilitiy is provided for inspecting collected traces or convert traces from SysFlow's compact binary format into human-readable JSON or CSV formats.

```
docker run --rm -v /mnt/data:/mnt/data sysflowtelemetry/sysprint /mnt/data/<trace>
```

where `trace` is the the name of the trace file inside `/mnt/data`. If empty, all files in `/mnt/data` are processed. By default, the traces are printed to standard output with a default set of SysFlow attributes. For a complete list of options, run:

```
docker run --rm -v /mnt/data:/mnt/data sysflowtelemetry/sysprint  -h
```

3.2 SysFlow Specification

SysFlow is an open specification for system event-level telemetry. The main goal of SysFlow is to create a standard and extensible data format for both security and performance analytics for compute workloads. An open standard will enable researchers and practitioners to more easily work on a common data format, and focus on analytics using open source software.

The primary objective of SysFlow is to lift raw system call data into more semantic process behaviors which promote significant data reductions for longer term forensic storage of data which is crucial for security analyzes. Through an object relational model of entities, events and flows, we enable SysFlow users to configure the desired granularity of data collection and filtering in order to facilitate most types of analysis in big data frameworks.

- *Overview*
- *Entities*
  - *Header*
  - *Container*
  - *Process*
  - *File*
- *Events*
  - *Operation Flags*
  - *Process Event*
  - *File Event*
  - *Network Event*
- *Flows*
  - *Process Flow*
  - *File Flow*
  - *Network Flow*

3.2.1 Overview

SysFlow is an object relational model of entities, events and flows that describe the behaviors of processes on a system, and encode them into an open format. A SysFlow exporter is designed to monitor system events of a workload, convert

them to SysFlow objects, and output them in a binary output file. We envision that one exporter will be deployed per host (or Virtual Machine) and will output one binary file over a particular time period. Figure 1 show a detailed view of the objects that the SysFlow exporter will export.

Entities represent the components on a system that we are interested in monitoring. In this version of SysFlow, we support three types of entities: Containers, Processes, and Files. As shown in Figure 1, Containers contain both Processes and Files, and the three are linked through object identifiers (more on this later).

Entity behaviors are modeled as events or flows. Events represent important individual behaviors of an entity that are broken out on their own due to their importance, their rarity, or because maintaining operation order is important. An example of an event would be a process clone or exec, or the deletion or renaming of a file. By contrast, a Flow represents an aggregation of multiple events that naturally fit together to describe a particular behavior. For example, we can model the network interactions of a process and a remote host as a bidirectional flow that is composed of several events, including connect, read, write, and close.

The idea behind SysFlow is to enable the user to configure the granularity of system-level data desired based on resource limitations and data analytics requirements. In this way, behaviors can be broken out into individual events or combined into smaller aggregated volumetric flows. The current version of the specification describes events and flows in three key behavioral areas: Files, Networks, and Processes. Figure 1 shows these events and flows with their attributes and relationships to entities, which are described in greater details in the following sections.

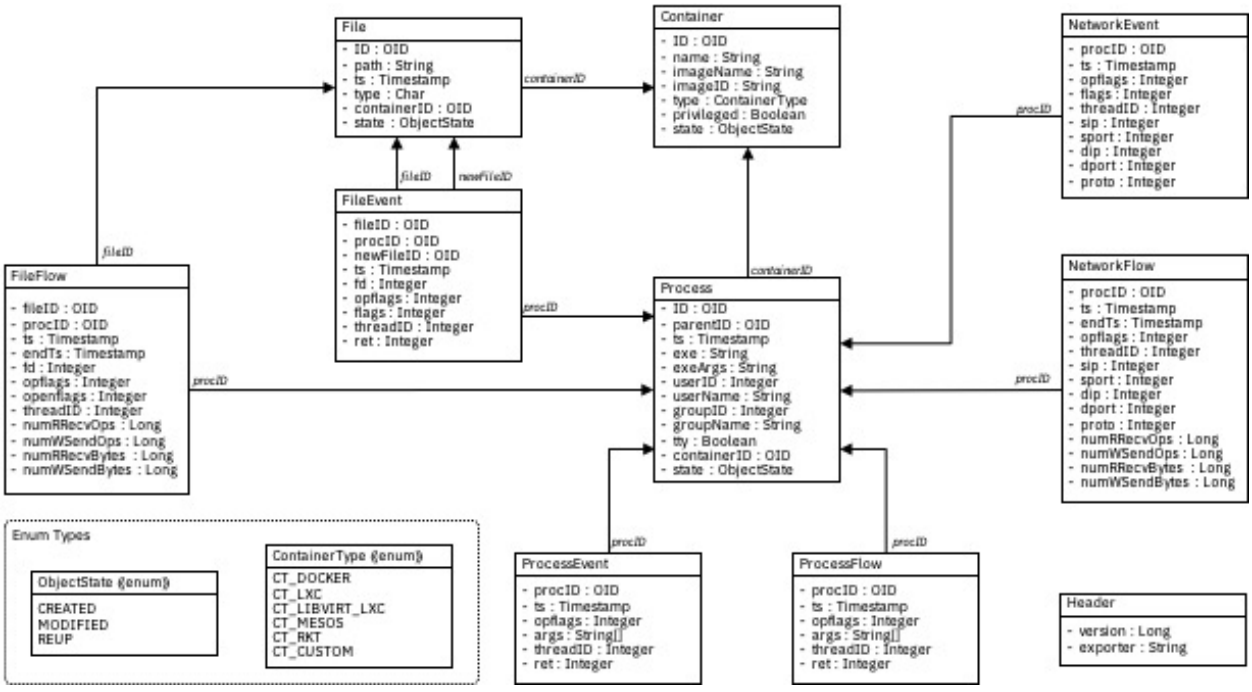

Figure 1: SysFlow Object Relational View

Entities

As mentioned above, entities are the components on a system that we are interested in monitoring. These include containers, processes, and files. We also support a special entity object called a Header, which stores information about the SysFlow version, and a unique ID representing the host or virtual machine monitored by the SysFlow exporter. The header is always the first record appearing in a SysFlow File. All other entities contain a timestamp, an object ID and a state. The timestamp is used to indicate the time at which the entity was exported to the SysFlow file.

Object ID

Object IDs allow events and flows to reference entities without having duplicate information stored in each record. Object IDs are not required to be globally unique across space and time. In fact, the only requirement for uniqueness is that no two objects managed by a SysFlow exporter can have the same ID simultaneously. Entities are always written to the binary output file before any events, and flows associated with them are exported. Since entities are exported first, each event, and flow is matched with the entity (with the same id) that is closest to it in the file. Furthermore, every binary output file must be self-contained, meaning that all entities referenced by flows/events must be present in every SysFlow file generated.

State

The state is an enumeration that indicates why an entity was written to disk. The state can currently be one of three values:

State	Description
CREATED	Indicates that the entity was recently created on the host/VM. For example, a process clone.
MODIFIED	Indicates that some attributes of the entity were modified since the last time it was exported.
REUP	Indicates that the entity already existed, but is being exported again, so that output files can be self-contained.

Each entity is defined below with recommendations on what to use for object identifiers, based on what is used in the current implementation of the SysFlow exporter.

Header

The Header entity is an object which appears at the beginning of each binary SysFlow file. It contains the current version of SysFlow as supported in the file, and the exporter ID.

Attribute	Type	Description	Since (schema version)
version	long	The current SysFlow version.	1
exporter	string	Globally unique id representing the host monitored by SysFlow.	1
ip	string	IP address in dot notation representing the monitored host.	2

Container

The Container entity represents a system or application container such as docker or LXC. It contains important information about the container including its id, name, and whether it is privileged.

Attribute	Type	Description	Since (schema version)
id	string	Unique string representing the Container Object as provided by docker, LXC, etc.	1
state	enum	state of the process (CREATED, MODIFIED, REUP).	not implemented
timestamp (ts)	int64	The timestamp when container object is exported (nanoseconds).	not implemented
name	string	Container name as provided by docker, LXC, etc.	1
image	string	Image name associated with container as provided by docker, LXC, etc.	1
imageID	string	Image ID associated with container as provided by docker, LXC, etc.	1
type	enum	Can be one of: CT_DOCKER, CT_LXC, CT_LIBVIRT_LXC, CT_MESOS, CT_RKT, CT_CUSTOM	1
privileged	boolean	If true, the container is running with root privileges	1

Process

The process entity represents a running process on the system. It contains important information about the process including its host pid, creation time, oid id, as well as references to its parent id. When a process entity is exported to a SysFlow file, all its parent processes should be exported before the process, as well as the process's Container entity. Processes are only exported to a SysFlow file if an event or flow associated with that process or any of its threads are exported. Threads are not explicitly exported in the process object but are represented in events and flows through a thread id field. Finally, a Process entity only needs to be exported to a file once, unless it's been modified by an event or flow.

> **NOTE:** In current implementation, the creation timestamp is the time at which the process is cloned. If the process was cloned before capture was started, this value is 0. The current implementation also has problems getting absolute paths for exes when relative paths are used to launch processes.

Attribute	Type	Description	Since (schema version)
state	enum	state of the process (CREATED, MODIFIED, REUP)	1
OID: *host pidcreate ts*	**struct** *int64int64*	The Process OID contains the host pid of the project, and creation timestamp.	1
POID: *parent host pidparent create ts*	**struct** *int64int64*	The OID of the parent process can be NULL if not available or if a root process.	1
timestamp (ts)	int64	The timestamp when process object is exported (nanoseconds).	1
exe	string	Full path (if available) of the executable used in the process launch; otherwise, it's the name of the exe.	1
exeArgs	string	Concatenated list of args passed on process startup.	1
uid	int32	User ID under which the process is running.	1
userName	string	User name under which the process is running.	1
gid	int32	Group ID under which the process is running	1
groupName	string	Group Name under which the process is running	1
tty	boolean	If true, the process is tied to a shell	1
containerId	string	Unique string representing the Container Object to which the process resides. It can be NULL if process isn't in a container.	1
entry	boolean	If true, the process is a container or system entrypoint (i.e., virtual pid = 1).	2

File

The File entity represents file-based resources on a system including files, directories, unix sockets, and pipes.

> **NOTE:** Current implementation does not have access to inode related values, which would greatly improve object ids. Also, the current implementation has some issues with absolute paths when monitoring operations that use relative paths.

Attribute	Type	Description	Since (schema version)
state	enum	state of the file (CREATED, MODIFIED, REUP)	1
FOID:	string (128bit)	File Identifier, is a SHA1 hash of the concatenation of the path + container ID	1
timestamp (ts)	int64	The timestamp when file object is exported (nanoseconds).	1
restype	enum	Indicates the resource type. Currently support: SF_FILE, SF_DIR, SF_UNIX (unix socket), SF_PIPE, SF_UNKNOWN	1
path	string	Full path of the file/directory, or unique identifier for pipe, unix socket	1
containerId	string	Unique string representing the Container Object to which the file resides. Can be NULL if file isn't in a container.	1

Events

Events represent important individual behaviors of an entity that are broken out on their own due to their importance, their rarity, or because maintaining operation order is important. In order to manage events and their differing attributes, we divide them into three different categories: Process, File, and Network events. These are described more in detail later on.

Each event and flow contains a process object id, a timestamp, a thread id, and a set of operation flags. The process object id represents the Process Entity on which the event occurred, while the thread id indicates which process thread was associated with the event.

Operation Flags

The operation flags describe the actual behavior associated with the event (or flow). The flags are represented in a single bitmap which enables multiple behaviors to be combined easily into a flow. An event will have a single bit active, while a flow could have several. The current supported flags are as follows:

Operation	Numeric ID	Description	System Calls	Evts/Flows Supported	Since (schema version)
OP_CLONE	(1 << 0)	Process or thread cloned.	clone()	ProcessEvent	1
OP_EXEC	(1 << 1)	Execution of a file	execve()	ProcessEvent	1
OP_EXIT	(1 << 2)	Process or thread exit.	exit()	ProcessEvent	1
OP_SETUID	(1 << 3)	UID of process was changed	setuid(), setresuid	ProcessEvent	1
OP_SETNS	(1 << 4)	Process entering namespace	setns()	FileFlow	1
OP_ACCEPT	(1 << 5)	Process accepting network connections	accept(), select()	NetworkFlow	1
OP_CONNECT	(1 << 6)	Process connecting to remote host or process	connect()	NetworkFlow	1
OP_OPEN	(1 << 7)	Process opening a file/resource	open(), openat(), create()	FileFlow	1
OP_READ_RECV	(1 << 8)	Process reading from file, receiving network data	read(),pread(),recv(),recvfrom(),recvmsg()	NetworkFlow, FileFlow	1
OP_WRITE_SEND	(1 << 9)	Process writing to file, sending network data	write(),pwrite(),send(),sendto(),sendmsg()	NetworkFlow, FileFlow	1
OP_CLOSE	(1 << 10)	Process close resource	close(),socketshutdown	NetworkFlow, FileFlow	1
OP_TRUNCATE	(1 << 11)	Premature closing of a flow due to exporter shutdown	N/A	NetworkFlow, FileFlow	1
OP_SHUTDOWN	(1 << 12)	Shutdown all or part of a full duplex socket connection	shutdown()	NetworkFlow	1
OP_MMAP	(1 << 13)	Memory map of a file.	mmap()	FileFlow	1
OP_DIGEST	(1 << 14)	Summary flow information for long running flows	N/A	NetworkFlow, FileFlow	1
OP_MKDIR	(1 << 15)	Make directory	mkdir(), mkdirat()	FileEvent	1
OP_RMDIR	(1 << 16)	Remove directory	rmdir()	FileEvent	1
OP_LINK	(1 << 17)	Process creates hard link to existing file	link(), linkat()	FileEvent	1
OP_UNLINK	(1 << 18)	Process deletes file	unlink(), unlinkat()	FileEvent	1
OP_SYMLINK	(1 << 19)	Process creates sym link to existing file	symlink(), symlinkat()	FileEvent	1
OP_RENAME	(1 << 20)	File renamed	rename(), renameat()	FileEvent	1

Process Event

A Process Event is an event that creates or modifies a process in some way. Currently, we support four Process Events (referred to as operations), and their behavior in SysFlow is described below.

Operation	Behavior
OP_CLONE	Exported when a new process or thread is cloned. A new Process Entity should be exported prior to exporting the clone operation of a new process.
OP_EXEC	Exported when a process calls an exec syscall. This event will modify an existing process, and should be accompanied by a modified Process Entity.
OP_EXIT	Exported on a process or thread exit.
OP_SETUID	Exported when a process's UID is changed. This event will modify an existing process, and should be accompanied by a modified Process Entity.

The list of attributes for the Process Event are as follows:

Attribute	Type	Description	Since (schema version)
OID: *host pidcreate ts*	**struct** *int64int64*	The OID of the process for which the event occurred.	1
timestamp (ts)	int64	The timestamp when the event occurred (nanoseconds).	1
tid	int64	The id of the thread associated with the ProcessEvent. If the running process is single threaded tid == pid	1
opFlags	int64	The id of the syscall associated with the event. See list of Operation Flags for details.	1
args	string[]	An array of arguments encoded as string for the syscall.	Sparingly implemented. Only really used with setuid for now.
ret	int64	Syscall return value.	1

File Event

A File Event is an event that creates, deletes or modifies a File Entity. Currently, we support six File Events (referred to as operations), and their behavior in SysFlow is described below.

Operation	Behavior
OP_MKDIR	Exported when a new directory is created. Should be accompanied by a new File Entity representing the directory
OP_RMDIR	Exported when a directory is deleted.
OP_LINK	Exported when a process creates a hard link to an existing file. Should be accompanied by a new File Entity representing the new link.
OP_UNLINK	Exported when a process deletes a file.
OP_SYMLINK	Exported when a process creates a sym link to an existing file. Should be accompanied by a new File Entity representing the new link.
OP_RENAME	Exported when a process creates renames an existing file. Should be accompanied by a new File Entity representing the renamed file.

> **NOTE:** We'd like to also support **chmod** and **chown** but these two operations are not fully supported in sysdig. We'd also like to support **umount** and **mount** but these operations are not implemented. We anticipate supporting these in a future version.

The list of attributes for the File Event are as follows:

Attribute	Type	Description	Since (schema version)
OID: *host pidcreatets*	**struct** *int64int64*	The OID of the process for which the event occurred.	1
timestamp (ts)	int64	The timestamp when the event occurred (nanoseconds).	1
tid	int64	The id of the thread associated with the FileEvent. If the running process is single threaded tid == pid	1
opFlags	int64	The id of the syscall associated with the event. See list of Operation Flags for details.	1
ret	int64	Syscall return value.	1
FOID:	string (128bit)	The id of the file on which the system call was called. File Identifier, is a SHA1 hash of the concatenation of the path + container ID.	1
NewFOID:	string (128bit)	Some syscalls (link, symlink, etc.) convert one file into another requiring two files. This id is the id of the file secondary or new file on which the system call was called. File Identifier, is a SHA1 hash of the concatenation of the path + container ID. Can be NULL.	1

Network Event

Currently, not implemented.

Flows

A Flow represents an aggregation of multiple events that naturally fit together to describe a particular behavior. They are designed to reduce data and collect statistics. Examples of flows include an application reading or writing to a file, or sending and receiving data from another process or host. Flows represent a number of events occurring over a period of time, and as such each flow has a set of operations (encoded in a bitmap), a start and an end time. One can determine the operations in the flow by decoding the operation flags.

A flow can be started by any supported operation and are exported in one of two ways. First, they are exported on an exit, or close event signifying the end of a connection, file interaction, or process. Second, a long running flow is exported after a preconfigured time period. After a long running flow is exported, its counters and flags are reset. However, if there is no activity on the flow over a preconfigured period of time, that flow is no longer exported.

In this section, we describe three categories of Flows: Process, File and Network Flows.

Process Flow

A Process Flow represents a summarization of the number of threads created and destroyed over a time period. Process Flows are partially implemented in the collector and will be fully implemented in a later release. Since schema version 2. Currently we support the following operations:

Operation	Behavior
OP_CLONE	Recorded when a new thread is cloned.
OP_EXIT	Recorded on a thread exit.

The list of attributes for the Process Flow are as follows:

Attribute	Type	Description	Since (schema version)
OID: *host pidcreate ts*	**struct** *int64int64*	The OID of the process for which the flow occurred.	2
timestamp (ts)	int64	The timestamp when the flow starts (nanoseconds).	2
numThreadsCloned	int64	The number of threads cloned during the duration of the flow.	2
opFlags	int64 (bitmap)	The id of one or more syscalls associated with the ProcessFlow. See list of Operation Flags for details.	2
endTs	int64	The timestamp when the process flow is exported (nanoseconds).	2
numThreadsExited	int64	Number of threads exited during the duration of the flow.	2
numCloneErrors	int64	Number of clone errors occuring during the duration of the flow.	2

File Flow

A File Flow represents a collection of operations on a file. Currently we support the following operations:

Operation	Behavior
OP_SETNS	Process entering namespace entry in mounted file related to reference File Entity
OP_OPEN	Process opening a file/resource.
OP_READ_RECV	Process reading from file/resource.
OP_WRITE_SEND	Process writing to file.
OP_MMAP	Processing memory mapping a file.
OP_CLOSE	Process closing resource. This action will close corresponding FileFlow.
OP_TRUNCATE	Indicates Premature closing of a flow due to exporter shutdown.
OP_DIGEST	Summary flow information for long running flows (not implemented).

The list of attributes for the File Flow are as follows:

Attribute	Type	Description	Since (schema version)
OID: *host pidcreate ts*	**struct** *int64int64*	The OID of the process for which the flow occurred.	1
timestamp (ts)	int64	The timestamp when the flow starts (nanoseconds).	1
tid	int64	The id of the thread associated with the flow. If the running process is single threaded tid == pid	1
opFlags	int64 (bitmap)	The id of one or more syscalls associated with the FileFlow. See list of Operation Flags for details.	1
openFlags	int64	Flags associated with an open syscall if present.	1
endTs	int64	The timestamp when the file flow is exported (nanoseconds).	1
FOID:	string (128bit)	The id of the file on which the system call was called. File Identifier, is a SHA1 hash of the concatenation of the path + container ID.	1
fd	int32	The file descriptor associated with the flow.	1
numRRecvOps	int64	Number of read operations performed during the duration of the flow.	1
numWSendOps	int64	Number of write operations performed during the duration of the flow.	1
numRRecvBytes	int64	Number of bytes read during the duration of the flow.	1
numWSendBytes	int64	Number of bytes written during the duration of the flow.	1

Network Flow

A Network Flow represents a collection of operations on a network connection. Currently we support the following operations:

Operation	Behavior
OP_ACCEPT	Process accepted a new network connection.
OP_CONNECT	Process connected to a remote host or process.
OP_READ_RECV	Process receiving data from a remote host or process.
OP_WRITE_SEND	Process sending data to a remote host or process.
OP_SHUTDOWN	Process shutdown full or single duplex connections.
OP_CLOSE	Process closing network connection. This action will close corresponding NetworkFlow.
OP_TRUNCATE	Indicates Premature closing of a flow due to exporter shutdown.
OP_DIGEST	Summary flow information for long running flows (not implemented).

The list of attributes for the Network Flow are as follows:

Attribute	Type	Description	Since (schema version)
OID: *host pidcreate ts*	**struct** *int64int64*	The OID of the process for which the flow occurred.	1
timestamp (ts)	int64	The timestamp when the flow starts (nanoseconds).	1
tid	int64	The id of the thread associated with the flow. If the running process is single threaded tid == pid	1
opFlags	int64 (bitmap)	The id of one or more syscalls associated with the flow. See list of Operation Flags for details.	1
endTs	int64	The timestamp when the flow is exported (nanoseconds).	1
sip	int32	The source IP address.	1
sport	int16	The source port.	1
dip	int32	The destination IP address.	1
dport	int16	The destination port.	1
proto	enum	The network protocol of the flow. Can be: TCP, UDP, ICMP, RAW	1
numRRecvOps	int64	Number of receive operations performed during the duration of the flow.	1
numWSendOps	int64	Number of send operations performed during the duration of the flow.	1
numRRecvBytes	int64	Number of bytes received during the duration of the flow.	1
numWSendBytes	int64	Number of bytes sent during the duration of the flow.	1

NOTE: The current implementation of NetworkFlow only supports ipv4.

3.3 SysFlow Collector (sf-collector repo)

The SysFlow Collector monitors and collects system call and event information from hosts and exports them in the SysFlow format using Apache Avro object serialization. SysFlow lifts system call information into a higher order object relational form that models how containers, processes and files interact with their environment through process control flow, file, and network operations. Learn more about SysFlow in the SysFlow Specification Document.

The SysFlow Collector is currently built upon a Sysdig core and requires the Sysdig probe to passively collect system events and turn them into SysFlow. As a result, the collector supports Sysdig's powerful filtering capabilities. Please see the build and installation instructions for installing the collector.

3.3.1 Installation and Usage

Installing the collector

Cloning source

The sf-collector project has been tested primarily on Ubuntu 16.04 and 18.04. The project will be tested on other flavors of UNIX in the future. This document describes how to build and run the application both inside a docker container and on a linux host. Building and running the application inside a docker container is the easiest way to start. For convenience, skip the build step and pull pre-built images directly from Docker Hub.

To build the project, first pull down the source code, with submodules:

```
git clone --recursive git@github.com:sysflow-telemetry/sf-collector.git
```

To checkout submodules on an already cloned repo:

```
git submodule update --init --recursive
```

Building as Docker container

To build as docker container:

```
cd sf-collector
make -C modules init
docker build --target runtime -t sf-collector .
```

The container is built in stages to enable caching of the intermediate steps of the build and reduce final image sizes.

Building directly on a host

First, install required dependencies:

```
apt install patch base-files binutils bzip2 libdpkg-perl perl make xz-utils␣
↪libncurses5-dev libncursesw5-dev cmake libboost-all-dev g++  flex bison wget libelf-
↪dev liblog4cxx-dev libapr1 libaprutil1 libsparsehash-dev
```

To build the collector:

```
cd sf-collector
make install
```

Running the collector

Running the collector from the command line

The collector has the following options:

```
Usage: sysporter [options] -w <file name/dir>

Options:
  -h                            Show this help message and exit
  -w file name/dir (required)   The file or directory to which sysflow records are␣
↪written. If a directory is specified     (using a trailing slash), file name will␣
↪be an epoch timestamp. If -G is specified, then the file name specified will have␣
↪an epoch timestamp appended to it
  -e exporterID                 A globally unique ID representing the host or VM␣
↪being monitored which is stored in the sysflow dumpfile header. If -e not set, the␣
↪hostname of the CURRENT machine is used, which may not be accurate for reading␣
↪offline scap files
  -G interval (in secs)         Rotates the dumpfile specified in -w every interval␣
↪seconds and appends epoch timestamp to file name
  -r scap file                  The scap file to be read and dumped as sysflow format␣
↪at the file specified by -w. If this option is not specified, a live capture is␣
↪assumed
```

(continues on next page)

(continued from previous page)

```
  -s schema file                The sysflow avro schema file (.avsc) used for schema␣
↪validation (default: /usr/local/sysflow/conf/SysFlow.avsc)
  -f filter                     Sysdig style filtering string to filter scap. Must be␣
↪surrounded by quotes
  -c                            Simple, fast filter to allow only container-related␣
↪events to be dumped
  -p cri-o path                 The path to the cri-o domain socket
  -t cri-o timeout              The amount of time in ms to wait for cri-o socket to␣
↪respond
  -u domain socket file         Outputs SysFlow to a unix domain socket rather than␣
↪to a file
  -v                            Print version information and exit
```

Example usage

Convert Sysdig scap file to SysFlow file with an export id. The output will be written to `output.sf`. Note that the collector must be run with root privilege:

```
sysporter -r input.scap -w ./output.sf  -e host
```

Trace a system live, and output SysFlow to files in a directory which are rotated every 30 seconds. The file name will be an epoch timestamp of when the file was initially written. Note that the trailing slash *must be present*. The example filter ensures that only SysFlow from containers is generated.

```
sysporter -G 30 -w ./output/ -e host -f "container.type!=host and container.
↪type=docker"
```

Trace a system live, and output SysFlow to files in a directory which are rotated every 30 seconds. The file name will be an `output.<epoch timestamp>` where the timestamp is of when the file was initially written. The example filter ensures that only SysFlow from containers is generated.

```
sysporter -G 30 -w ./output/output -e host -f "container.type!=host and container.
↪type=docker" </code>`
```

Running the collector from a Docker container

The easiest way to run the SysFlow collector is from a Docker container, with host mount for the output trace files. The following command shows how to run sf-collector with trace files exported to `/mnt/data` on the host.

```
docker run -d --privileged --name sf-collector \
         -v /var/run/docker.sock:/host/var/run/docker.sock \
         -v /dev:/host/dev -v /proc:/host/proc:ro \
         -v /boot:/host/boot:ro -v /lib/modules:/host/lib/modules:ro \
             -v /usr:/host/usr:ro -v /mnt/data:/mnt/data \
             -e INTERVAL=60 \
             -e EXPORTER_ID=${HOSTNAME} \
             -e OUTPUT=/mnt/data/    \
             -e FILTER="container.name!=sf-collector and container.name!=sf-exporter"␣
↪\
             --rm sysflow-telemetry/sf-collector
```

where INTERVAL denotes the time in seconds before a new trace file is generated, EXPORTER_ID sets the exporter name, OUTPUT is the directory in which trace files are written, and FILTER is the filter expression used to filter

collected events. Note: append `container.type!=host` to FILTER expression to filter host events.

CRI-O support

The sf-collector project currently supports docker and kubernetes deployments (using the helm charts provided in sf-deployments). Container runtimes based on CRI-O is planned for futures releases of the collector.

3.4 SysFlow Exporter (sf-exporter repo)

SysFlow exporter to export SysFlow traces to S3-compliant object stores and rsyslog servers.

3.4.1 Cloning source

The sf-exporter project has been tested primarily on Ubuntu 16.04 and 18.04. The project will be tested on other flavors of UNIX in the future. This document describes how to build and run the application both inside a docker container and on a Linux host. Building and running the application inside a docker container is the easiest way to start. For convenience, skip the build step and pull pre-built images directly from Docker Hub.

To build the project, first pull down the source code, with submodules:

```
git clone --recursive git@github.com:sysflow-telemetry/sf-exporter.git
```

To checkout submodules on an already cloned repo:

```
git submodule update --init --recursive
```

3.4.2 Container

```
docker build --pull --force-rm -t sf-exporter .
```

For s3 export:

```
docker service create --name sf-exporter \
    -e NODE_IP=10.1.0.159 \
    -e INTERVAL=15 \
    --secret s3_access_key \
    --secret s3_secret_key \
    --mount type=bind,source=/mnt/data,destination=/mnt/data \
    sf-exporter:latest
```

For remote syslogging:

```
docker service create --name sf-exporter \
    -e SYSLOG_HOST=localhost \
    -e SYSLOG_PORT=514 \
    -e NODE_IP=10.1.0.159 \
    -e INTERVAL=15 \
    -e DIR=/mnt/data \
    --mount type=bind,source=/mnt/data,destination=/mnt/data \
    sf-exporter:latest
```

3.4.3 Development

```
cd src & pip3 install -r requirements.txt
cd modules/sysflow/py3 & sudo python3 setup.py install
```

Example run with remote syslogging export:

```
./exporter.py --exporttype syslog --sysloghost 127.0.0.1 --syslogport 514 --dir /mnt/
↪data --nodeip testnode --scaninterval 15
```

3.4.4 Usage

```
usage: exporter.py [-h] [--exporttype {s3,syslog}] [--sysloghost SYSLOGHOST]
                   [--syslogport SYSLOGPORT] [--s3endpoint S3ENDPOINT]
                   [--s3port S3PORT] [--s3accesskey S3ACCESSKEY]
                   [--s3secretkey S3SECRETKEY] [--secure [SECURE]]
                   [--scaninterval SCANINTERVAL] [--timeout TIMEOUT]
                   [--agemin AGEMIN] [--dir DIR] [--s3bucket S3BUCKET]
                   [--s3location S3LOCATION] [--nodename NODENAME]
                   [--nodeip NODEIP] [--podname PODNAME] [--podip PODIP]
                   [--podservice PODSERVICE] [--podns PODNS]
                   [--poduuid PODUUID]

sf-exporter: service for watching and uploading monitoring files to object
store.

optional arguments:
  -h, --help            show this help message and exit
  --exporttype {s3,syslog}
                        export type
  --sysloghost SYSLOGHOST
                        syslog host address
  --syslogport SYSLOGPORT
                        syslog UDP port
  --s3endpoint S3ENDPOINT
                        s3 server address
  --s3port S3PORT     s3 server port
  --s3accesskey S3ACCESSKEY
                        s3 access key
  --s3secretkey S3SECRETKEY
                        s3 secret key
  --secure [SECURE]     indicates if SSL connection
  --scaninterval SCANINTERVAL
                        interval between scans
  --timeout TIMEOUT     connection timeout
  --agemin AGEMIN       number of minutes of traces to preserve in case of
                        repeated timeouts
  --dir DIR             data directory
  --s3bucket S3BUCKET
                        target data bucket
  --s3location S3LOCATION
                        target data bucket location
  --nodename NODENAME   exporter's node name
  --nodeip NODEIP       exporter's node IP
  --podname PODNAME     exporter's pod name
```

(continues on next page)

(continued from previous page)

```
  --podip PODIP         exporter's pod IP
  --podservice PODSERVICE
                        exporter's pod service
  --podns PODNS         exporter's pod namespace
  --poduuid PODUUID     exporter's pod UUID
```

3.5 SysFlow APIs and Utilities (sf-apis repo)

3.5.1 SysFlow APIs and Utilities

SysFlow uses Apache Avro serialization to create compact records that can be processed by a wide variety of programming languages, and big data analytics platforms such as Apache Spark. Avro enables a user to generate programming stubs for serializing and deserializing data, using either Apache Avro IDL or Apache schema files.

Cloning source

The sf-apis project has been tested primarily on Ubuntu 16.04 and 18.04. The project will be tested on other flavors of UNIX in the future. This document describes how to build and run the application both on a linux host.

To build the project, first pull down the source code:

```
git clone git@github.com:sysflow-telemetry/sf-apis.git
```

Avro IDL and schema files

The Avro IDL files for SysFlow are available in the repository under `sf-apis/avro/avdl`, while the schema files are available under `sf-apis/avro/avsc`. The `avrogen` tool can be used to generate classes using the schema. See `sf-apis/avro/generateCClasses.sh` for an example of how to generate C++ headers from apache schema files.

SysFlow Avro C++

SysFlow C++ SysFlow objects and encoders/decoders are all available in `sf-apis/c++/sysflow/sysflow.hh`. `sf-collector/src/sysreader.cpp` provides a good example of how to read and process different SysFlow avro objects in C++. Note that one must install Apache Avro 1.9.1 cpp to run an application that includes `sysflow.hh`. The library file `-lavrocpp` must also be linked during compilation.

SysFlow Avro Python 3

SysFlow Python 3 APIs are generated with the avro-gen Python package. These classes are available in `sf-apis/py3`.

In order to install the SysFlow Python package:

```
cd sf-apis/py3
sudo python3 setup.py install
```

Please see the SysFlow Python API reference documents for more information on the modules and objects in the library.

SysFlow utilities

sysprint

`sysprint` is a tool written using the SysFlow Python API that will print out SysFlow traces from a file into several different formats including JSON, CSV, and tabular pretty print form. Not only will sysprint help you interact with SysFlow, it is also a good example for how to write new analytics tools using the SysFlow API.

```
usage: sysprint [-h] [-i {local,s3}] [-o {str,json,csv}] [-w FILE]
                [-f FIELDS] [-c S3ENDPOINT] [-p S3PORT] [-a S3ACCESSKEY]
                [-s S3SECRETKEY] [-l S3LOCATION] [--secure [SECURE]]
                path [path ...]

sysprint: a human-readable printer for Sysflow captures.

positional arguments:
  path                  list of paths or bucket names from where to read trace
                        files

optional arguments:
  -h, --help            show this help message and exit
  -i {local,s3}, --input {local,s3}
                        input type
  -o {str,json,csv}, --output {str,json,csv}
                        output format
  -w FILE, --file FILE  output file path
  -f FIELDS, --fields FIELDS
                        comma-separated list of sysflow fields to be printed
  -c S3ENDPOINT, --s3endpoint S3ENDPOINT
                        s3 server address from where to read sysflows
  -p S3PORT, --s3port S3PORT
                        s3 server port
  -a S3ACCESSKEY, --s3accesskey S3ACCESSKEY
                        s3 access key
  -s S3SECRETKEY, --s3secretkey S3SECRETKEY
                        s3 secret key
  -l S3LOCATION, --s3location S3LOCATION
                        target data bucket location
  --secure [SECURE]     indicates if SSL connection
```

3.5.2 SysFlow Python API Reference

SysFlow Reader API

class sysflow.reader.**FlattenedSFReader**(*filename*, *retEntities=False*)

FlattenedSFReader

This class loads a raw sysflow file, and links all Entities (header, process, container, files) with the current flow or event in the file. As a result, the user does not have to manage this information. This class supports the python iterator design pattern. Example Usage:

```
reader = FlattenedSFReader(trace)
head = 20 # max number of records to print
for i, (objtype, header, cont, pproc, proc, files, evt, flow) in␣
↪enumerate(reader):
```

(continues on next page)

(continued from previous page)

```
    exe = proc.exe
    pid = proc.oid.hpid if proc else ''
    evflow = evt or flow
    tid = evflow.tid if evflow else ''
    opFlags = utils.getOpFlagsStr(evflow.opFlags) if evflow else ''
    sTime = utils.getTimeStr(evflow.ts) if evflow else ''
    eTime = utils.getTimeStr(evflow.endTs) if flow else ''
    ret = evflow.ret if evt else ''
    res1 = ''
    if objtype == ObjectTypes.FILE_FLOW or objtype == ObjectTypes.FILE_EVT:
        res1 = files[0].path
    elif objtype == ObjectTypes.NET_FLOW:
        res1 = utils.getNetFlowStr(flow)
    numBReads = evflow.numRRecvBytes if flow else ''
    numBWrites = evflow.numWSendBytes if flow else ''
    res2 = files[1].path if files and files[1] else ''
    cont = cont.id if cont else ''
    print("|{0:30}|{1:9}|{2:26}|{3:26}|{4:30}|{5:8}|{6:8}|".format(exe, opFlags,␣
↪sTime, eTime, res1, numBReads, numBWrites))
    if i == head:
        break
```

Parameters

- **filename** (*str*) – the name of the sysflow file to be read.
- **retEntities** (*bool*) – If True, the reader will return entity objects by themselves as they are seen in the sysflow file. In this case, all other objects will be set to None

Iterator Reader returns a tuple of objects in the following order:

objtype (*sysflow.objtypes.ObjectTypes*) The type of entity or flow returned.

header (`sysflow.entity.SFHeader`) The header entity of the file.

cont (`sysflow.entity.Container`) The container associated with the flow/evt, or None if no container.

pproc (`sysflow.entity.Process`) The parent process associated with the flow/evt.

proc (`sysflow.entity.Process`) The process associated with the flow/evt.

files (tuple of `sysflow.entity.File`) Any files associated with the flow/evt.

evt (`sysflow.event.{ProcessEvent,FileEvent}`) If the record is an event, it will be returned here. Otherwise this variable will be None. objtype will indicate the type of event.

flow (`sysflow.flow.{NetworkFlow,FileFlow}`) If the record is a flow, it will be returned here. Otherwise this variable will be None. objtype will indicate the type of flow.

getProcess(*oid*)

Returns a Process Object given a process object id.

Parameters **oid** (*sysflow.type.OID*) – the object id of the Process Object requested

Return type sysflow.entity.Process

Returns the desired process object or None if no process object is available.

class sysflow.reader.**NestedNamespace**(***kwargs*)

class sysflow.reader.**SFReader**(*filename*)

SFReader

This class loads a raw sysflow file, and returns each entity/flow one by one. It is the user's responsibility to link the related objects together through the OID. This class supports the python iterator design pattern. Example Usage:

```
reader = SFReader("./sysflowfile.sf")
for name, sf in reader:
    if name == "sysflow.entity.SFHeader":
       //do something with the header object
    elif name == "sysflow.entity.Container":
       //do something with the container object
    elif name == "sysflow.entity.Process":
       //do something with the Process object
    ....
```

Parameters **filename** (*str*) – the name of the sysflow file to be read.

SysFlow Formatter API

class sysflow.formatter.**SFFormatter**(*reader*, *defs=[]*)

SFFormatter

This class takes a *FlattenedSFReader*, and exports SysFlow as either JSON, CSV or Pretty Print . Example Usage:

```
reader = FlattenedSFReader(trace, False)
formatter = SFFormatter(reader)
fields=args.fields.split(',') if args.fields else None
if args.output == 'json':
    if args.file is not None:
        formatter.toJsonFile(args.file, fields=fields)
    else:
        formatter.toJsonStdOut(fields=fields)
elif args.output == 'csv' and args.file is not None:
    formatter.toCsvFile(args.file, fields=fields)
elif args.output == 'str':
    formatter.toStdOut(fields=fields)
```

Parameters

- **reader** (sysflow.reader.FlattenedSFReader) – A reader representing the sysflow file being read.
- **defs** (*list*) – A list of paths to filter definitions.

applyFuncJson(*func*, *fields=None*, *expr=None*)

Enables a delegate function to be applied to each JSON record read.

Parameters

- **func** (*function*) – delegate function of the form func(str)
- **fields** (*list*) – a list of the SysFlow fields to be exported in JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

getFields()
Returns a list with available SysFlow fields and their descriptions.

toCsvFile(*path*, *fields=None*, *header=True*, *expr=None*)
Writes SysFlow to CSV file.

Parameters

- **path** (*str*) – the full path of the output file.
- **fields** (*list*) – a list of the SysFlow fields to be exported in the JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

toDataframe(*fields=None*, *expr=None*)
Enables a delegate function to be applied to each JSON record read.

Parameters

- **func** (*function*) – delegate function of the form func(str)
- **fields** (*list*) – a list of the SysFlow fields to be exported in the JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

toJson(*fields=None*, *flat=False*, *expr=None*)
Writes SysFlow as JSON object.

Parameters

- **fields** (*list*) – a list of the SysFlow fields to be exported in JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

Flat specifies if JSON output should be flattened

toJsonFile(*path*, *fields=None*, *flat=False*, *expr=None*)
Writes SysFlow to JSON file.

Parameters

- **path** (*str*) – the full path of the output file.
- **fields** (*list*) – a list of the SysFlow fields to be exported in JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

Flat specifies if JSON output should be flattened

toJsonStdOut(*fields=None*, *flat=False*, *expr=None*)
Writes SysFlow as JSON to stdout.

Parameters

- **fields** (*list*) – a list of the SysFlow fields to be exported in JSON. See formatter.py for a list of fields
- **expr** (*str*) – a sfql filter expression

Flat specifies if JSON output should be flattened

toStdOut (*fields=['ts_uts', 'type', 'proc.exe', 'proc.args', 'pproc.pid', 'proc.pid', 'proc.tid', 'opflags', 'res', 'flow.rbytes', 'flow.wbytes', 'container.id'], pretty_headers=True, showindex=True, expr=None*)

Writes SysFlow as a tabular pretty print form to stdout.

Parameters

- **fields** (*list*) – a list of the SysFlow fields to be exported in the JSON. See formatter.py for a list of fields
- **pretty_headers** (*bool*) – print table headers in pretty format.
- **showindex** (*bool*) – show record number.
- **expr** (*str*) – a sfql filter expression

SysFlow Object Types

class sysflow.objtypes.**ObjectTypes**

ObjectTypes

Enumeration representing each of the object types: HEADER = 0, CONT = 1, PROC = 2, FILE = 3, PROC_EVT = 4, NET_FLOW = 5, FILE_FLOW = 6, FILE_EVT = 7

SysFlow Utils API

sysflow.utils.**getIpIntStr**(*ipInt*)

Converts an IP address in host order integer to a string representation.

Parameters **ipInt** – an IP address integer

Return type str

Returns A string representation of the IP address

sysflow.utils.**getNetFlowStr**(*nf*)

Converts a NetworkFlow into a string representation.

Parameters **nf** (sysflow.schema_classes.SchemaClasses.sysflow.flow.NetworkFlowClass) – a NetworkFlow object.

Return type str

Returns A string representation of the NetworkFlow in form (sip:sport-dip:dport).

sysflow.utils.**getOpFlags**(*opFlags*)

Converts a sysflow operations flag bitmap into a set representation.

Parameters **opflag** (*int*) – An operations bitmap from a flow or event.

Return type set

Returns A set representation of the operations bitmap.

sysflow.utils.**getOpFlagsStr**(*opFlags*)

Converts a sysflow operations flag bitmap into a string representation.

Parameters **opflag** (*int*) – An operations bitmap from a flow or event.

Return type str

Returns A string representation of the operations bitmap.

sysflow.utils.**getOpStr**(*opFlags*)
Converts a sysflow operations into a string representation.

Parameters **opflag** (*int*) – An operations bitmap from a flow or event.

Return type str

Returns A string representation of the operations bitmap.

sysflow.utils.**getOpenFlags**(*openFlags*)
Converts a sysflow open modes flag bitmap into a set representation.

Parameters **opflag** – An open modes bitmap from a flow or event.

Return type set

Returns A set representation of the open modes bitmap.

sysflow.utils.**getTimeStr**(*ts*)
Converts a nanosecond ts into a string representation.

Parameters **ts** (*int*) – A nanosecond epoch.

Return type str

Returns A string representation of the timestamp in %m/%d/%YT%H:%M:%S.%f format.

sysflow.utils.**getTimeStrIso8601**(*ts*)
Converts a nanosecond ts into a string representation in UTC time zone.

Parameters **ts** (*int*) – A nanosecond epoch.

Return type str

Returns A string representation of the timestamp in ISO 8601 format.

SysFlow Class

sysflow.sysflow.**SysFlow**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.SysFlowClass*

class sysflow.schema_classes.SchemaClasses.sysflow.**SysFlowClass**(*inner_dict=None*)

rec

Return type SchemaClasses.sysflow.entity.SFHeaderClass | SchemaClasses.sysflow.entity.ContainerClass | SchemaClasses.sysflow.entity.ProcessClass | SchemaClasses.sysflow.entity.FileClass | SchemaClasses.sysflow.event.ProcessEventClass | SchemaClasses.sysflow.flow.NetworkFlowClass | SchemaClasses.sysflow.flow.FileFlowClass | SchemaClasses.sysflow.event.FileEventClass | SchemaClasses.sysflow.event.NetworkEventClass | SchemaClasses.sysflow.flow.ProcessFlowClass

Container Class

sysflow.sysflow.entity.**Container**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.entity.ContainerClass*

class sysflow.schema_classes.SchemaClasses.sysflow.entity.**ContainerClass**(*inner_dict=None*)

id

Return type str

image

Return type str

imageid

Return type str

name

Return type str

privileged

Return type bool

type

Return type SchemaClasses.sysflow.type.ContainerTypeClass

File Class

sysflow.sysflow.entity.**File**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.entity.FileClass*

class sysflow.schema_classes.SchemaClasses.sysflow.entity.**FileClass**(*inner_dict=None*)

containerId

Return type str

oid

Return type bytes

path

Return type str

restype

Return type int

state

Return type SchemaClasses.sysflow.type.SFObjectStateClass

ts

Return type int

Header Class

sysflow.sysflow.entity.**SFHeader**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.entity.SFHeaderClass*

class sysflow.schema_classes.SchemaClasses.sysflow.entity.**SFHeaderClass**(*inner_dict=None*)

exporter

Return type str

ip

Return type str

version

Return type int

Process Class

sysflow.sysflow.entity.**Process**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.entity.ProcessClass*

class sysflow.schema_classes.SchemaClasses.sysflow.entity.**ProcessClass**(*inner_dict=None*)

containerId

Return type str

entry

Return type bool

exe

Return type str

exeArgs

Return type str

gid

Return type int

groupName

Return type str

oid

Return type SchemaClasses.sysflow.type.OIDClass

poid

Return type SchemaClasses.sysflow.type.OIDClass

state

Return type SchemaClasses.sysflow.type.SFObjectStateClass

ts

Return type int

tty

Return type bool

uid

Return type int

userName

Return type str

File Event

sysflow.sysflow.event.**FileEvent**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.event.FileEventClass*

class sysflow.schema_classes.SchemaClasses.sysflow.event.**FileEventClass**(*inner_dict=None*)

fileOID

Return type bytes

newFileOID

Return type bytes

opFlags

Return type int

procOID

Return type SchemaClasses.sysflow.type.OIDClass

ret

Return type int

tid

Return type int

ts

Return type int

Process Event

sysflow.sysflow.event.**ProcessEvent**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.event.ProcessEventClass*

class sysflow.schema_classes.SchemaClasses.sysflow.event.**ProcessEventClass**(*inner_dict=None*)

args

Return type list[str]

opFlags

Return type int

procOID

Return type SchemaClasses.sysflow.type.OIDClass

ret

Return type int

tid

Return type int

ts

Return type int

File Flow

sysflow.sysflow.flow.**FileFlow**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.flow.FileFlowClass*

class sysflow.schema_classes.SchemaClasses.sysflow.flow.**FileFlowClass**(*inner_dict=None*)

endTs

Return type int

fd

Return type int

fileOID

Return type bytes

numRRecvBytes

Return type int

numRRecvOps

Return type int

numWSendBytes

Return type int

numWSendOps

Return type int

opFlags

Return type int

openFlags

Return type int

procOID

Return type SchemaClasses.sysflow.type.OIDClass

tid

Return type int

ts

Return type int

Network Flow

sysflow.sysflow.flow.**NetworkFlow**
alias of *sysflow.schema_classes.SchemaClasses.sysflow.flow.NetworkFlowClass*

class sysflow.schema_classes.SchemaClasses.sysflow.flow.**NetworkFlowClass**(*inner_dict=None*)

dip

Return type int

dport

Return type int

endTs

Return type int

fd

Return type int

numRRecvBytes

Return type int

numRRecvOps

Return type int

numWSendBytes

Return type int

numWSendOps

Return type int

opFlags

Return type int

procOID

Return type SchemaClasses.sysflow.type.OIDClass

proto

Return type int

sip

Return type int

sport

Return type int

tid

Return type int

ts

Return type int

3.6 Deployments (sf-deployments repo)

This repository contains scripts for deploying the SysFlow telemetry pipeline in the following container environments:

- docker
- k8s
- OpenShift

3.6.1 Docker

Introduction

This repository contains utility scripts to deploy a docker telemetry stack.

Deployment

A *local collection* (Option 1) and a *full stack* (Option 2) deployment models are described below. The local deployment stores collected traces on the local filesystem and the full stack deployment exports the collected traces to a S3-compatible object storage server.

Prerequisites

To guarantee a smooth deployment, the kernel headers must be installed in the host operating system.

This can usually be done on Debian-like distributions with:

```
apt-get -y install linux-headers-$(uname -r)
```

Or, on RHEL-like distributions:

```
yum -y install kernel-devel-$(uname -r)
```

Setup

Clone the repository and navigate to this directory.

```
git clone git@github.com:sysflow-telemetry/sf-deployments.git
cd sf-deployments/docker
```

Option 1: Local telemetry deployment: Sysflow collection probe only

This deployment will install the Sysflow collection probe only, i.e., without an exporter to a data store (e.g., COS). See below for the deployment of the full telemetry stack.

Start collection probe

Start the telemetry probe, which will be ran in a container.

> Tip: add container.type!=host to FILTER string located inside this script to filter out host (non-containerized) events.

```
docker-compose -f docker-compose.collector.yml up
```

Stop collection probe

```
docker-compose -f docker-compose.collector.yml down
```

RSyslog exporter (optional)

If exporting to rsyslog (e.g., QRadar), specify the IP and port of the remote syslog server:

```
docker run --name sf-exporter \
    -e SYSLOG_HOST=<RSYSLOG IP> \
    -e SYSLOG_PORT=<RSYSLOG PORT> \
    -e NODE_IP=<EXPORTER HOST IP> \
    -e INTERVAL=15 \
    -e DIR=/mnt/data \
    -v /mnt/data:/mnt/data \
    sysflowtelemetry/sf-exporter:latest
```

Option 2: Full telemetry stack deployment: Sysflow collector probe and S3 exporter

Note: skip this if deploying locally.

Create docker secrets

Create the docker secrets used to connect to the object store:

```
echo "<s3 access key>" > ./secrets/access_key
echo "<s3 secret key>" > ./secrets/secret_key
```

Start telemetry stack (with local object store)

```
docker-compose -f docker-compose.minio.yml -f docker-compose.yml up
```

Stop telemetry stack (with local object store)

```
docker-compose -f docker-compose.minio.yml -f docker-compose.yml down
```

Start telemetry stack (external object store)

If exporting to a remote object store, modify the exporter settings in `docker-compose.yml`, then run:

```
docker-compose -f docker-compose.yml up
```

Stop telemetry stack (external object store)

```
docker-compose -f docker-compose.yml down
```

Sysflow trace inspection

Run `sysprint` and point it to a trace file. In the examples below, `sysprint` is an alias for:

```
docker run --rm -v /mnt/data:/mnt/data sysflowtelemetry/sysprint
```

Tabular output

```
sysprint /mnt/data/<trace name>
```

JSON output

```
sysprint -o json /mnt/data/<trace name>
```

CSV output

```
sysprint -o csv /mnt/data/<trace name>
```

Inspect traces exported to an object store:

```
sysprint -i s3 -c <s3_endpoint> -a <s3_access_key> -s <s3_secret_key> <bucket_name>
```

Tip: see all options of the `sysprint` utility with `-h` option.

Inspect example traces

Sample trace files are provided in `tests`. Copy them into `/mnt/data` to inspect inside sysprint's environment.

```
sysprint /mnt/data/tests/client-server/tcp-client-server.sf
```

Tip: other samples can be found in the tests directory

3.6.2 Helm Charts

The sf-deployments repository contains a set of helm charts (under the helm directory) used to deploy the sysflow collector, and exporter into a K8s environment. It also contains a test harness for testing the telemetry infrastructure against various workloads. Think of helm as a package manager for deploying kubernetes pods and services into a cloud. The charts tell Helm how to conduct these deployments and allow the user to change a wide array of configurations.

The SysFlow telemetry infrastructure is designed such that it should be deployable in any cloud environment. It has been tested on IBM Cloud, and as time permits, we will test it on other public/private cloud offerings. There are likely some minor differences in how the authentication works on each cloud which could require changes to these charts.

NOTE: This document has been tested with helm version 2.12, 2.16, and 3.1. Some helm commands may not work with other versions of helm. We've tested the framework on k8s versions 1.14, 1.15, 1.16.

Prerequisites

1. Install Helm: https://helm.sh/
2. Setup TLS for Helm: https://v2.helm.sh/docs/tiller_ssl/
3. S3 compliant object store - Currently tested with IBM's cloud object store, and minio object store (https://docs.min.io/).
    - Setup IBM Cloud Object store: https://cloud.ibm.com/docs/services/cloud-object-storage/iam?topic=cloud-object-storage-getting-started
4. Create Cloud Object Store HMAC Access ID, and Secret Key.
    - For IBM Cloud Object store: https://cloud.ibm.com/docs/services/cloud-object-storage/iam?topic=cloud-object-storage-service-credentials

Clone the Repository

First, clone the repo with the following command:

```
git clone git clone https://github.com/sysflow-telemetry/sf-deployments.git .
cd sf-deployments/helm
```

Helm - sf-exporter-chart

The sf-exporter-chart resides in the `sf-exporter-chart` folder. The exporter chart is a kubernetes daemonset, which deploys the sf-collector, and the sf-exporter to each node in the cluster. The sf-collector monitors the node, and writes sysflow to a shared mount `/mnt/data`. The sf-exporter reads from the `/mnt/data` and pushes completed files to an S3 compliant object store for analysis before deleting them.

An install script called `./installExporterChart` is provided to make using the helm chart easier. This script sets up the environment including k8s secrets. To use it, first go into the sf-exporter-chart directory and copy `values.yaml` to `values.yaml.local` and begin tailoring this yaml to your environment. Note that some of values set in here are passable through the installation script for safety reasons. Note that the install and delete scripts assume that tls is NOT installed. To support tls, simply add `--tls` to the helm commands at the bottom of each file.

```
registry:
  secretName: ""

# sysflow collection probe parameters
sfcollector:
  # image repository
  repository: sysflowtelemetry/sf-collector
  # image tag
  tag: latest
  # timeout in seconds to start roll a new trace files
  interval: 300
  # output directory, where traces are written to inside container
  outDir: /mnt/data/
  # collection filter
  filter: "\"container.type!=host and container.name!=sfexporter and container.name!
↪=sfcollector\""
  #Use this criPath if running docker runtime
  #criPath: ""
  #uUse this criPath if running containerd runtime
```

(continues on next page)

(continued from previous page)

```
  criPath: "/var/run/containerd/containerd.sock"
  #uUse this criPath if running crio runtime
  #criPath: "/var/run/crio/crio.sock"


# sysflow exporter parameters
sfexporter:
  # image repository
  repository: sysflowtelemetry/sf-exporter
  # image tag
  tag: latest
  # export inverval
  interval: 30
  # directory where traces are read from inside container
  outDir: /mnt/data/
  # object store address (overridden by install script)
  s3Endpoint: "<ip address>"
  # object store port
  s3Port: 443
  # object store bucket where to push traces
  s3Bucket: sysflow-bucket
  # object store location (overridden by install script)
  s3Location: us-south
  # object store access key (overridden by install script)
  s3AccessKey: "<s3_access_key>"
  # object store secret key (overridden by install script)
  s3SecretKey: "<s3_secret_key>"
  # object store connection, 'true' if TLS-enabled, 'false' otherwise
  s3Secure: "false"

nameOverride: "sfexporter"
fullnameOverride: ""
tmpfsSize: 500Mi # size of tmpfs shared volume between collector and exporter (where␣
↪traces are written)
```

Most of the defaults should work in any environment. The collector is currently set to rotating files in 5 min intervals (or 300 seconds). The `/mnt/data/` is mapped to a tmpfs filesystem, and you can specify its size using the `tmpfsSize`. CGroup resource limits can be set on the collector and exporter to limit resource usage. These can be adjusted depending on requirements and resources limitations.

For connecting to an S3 compliant data store, first take note of which port the S3 data store (`s3Port`) is configured. IBM Cloud COS listens on port 443, but certain minio installations can listen on port 9000. Also, if TLS is enabled on the S3 datastore, ensure `s3Secure` is `true`. Ensure that the `s3Bucket` is set to the desired S3 bucket location. The `s3Location`, `s3AccessKey` and `s3SecretKey` and `s3Endpoint` are each passed in through the installation script if you use it.

Kubernetes can use different container runtimes. Older versions used the docker runtime; however, newer versions typically run either containerd or crio. It's important to know which runtime you have if you want to get the full benefits of sysflow. You tell the collector which runtime you are using based on the sock file you refer too in the `criPath` variable. If you are using the `docker` runtime, leave `criPath` blank. If you are using containerd, set `criPath` to "/var/run/containerd/containerd.sock" and if you are using crio, set `criPath` to "/var/run/crio/crio.sock". If SysFlow files are empty or the container name variable is set to `incomplete` in SysFlow traces, this typically means that the runtime socket is not connected properly.

There are two versions of the install script. One in `sf-deployments/helm/helm-scpts-v2` works with helm version 2, while the other in `sf-deployments/helm/helm-scpts-v3` works with version 3. Run the script from the `helm` directory. For example:

```
./helm-scpts-v2/installExporterChart <s3_region> <s3_access_key> <s3_secret_key> <s3_
↪endpoint>
   <s3_region> value is the region in the S3 compliant object store (e.g., us-south)
   <s3_access_key> is the access key present in S3 compliant service credentials
   <s3_secret_key> is the secret key present in S3 compliant service credentials
   <s3_endpoint> is the address of the S3 compliant object store (e.g., s3.us-south.
↪cloud-object-storage.appdomain.cloud)
```

Note the install script installs the pods into a K8s namespace called `sysflow`

To check that the install worked, run:

```
kubectl get pods -n sysflow
```

To check the log output of the collector container in a pod:

```
kubectl logs -f -c sfcollector <podname>  -n sysflow
```

To check the log output of the exporter container in a pod:

```
kubectl logs -f -c sfexporter <podname>  -n sysflow
```

To delete the exporter chart run:

```
./helm-scpts-v2/deleteExporterChart
```

3.6.3 OpenShift Operator

This document describes how to use the Red Hat OpenShift (OC) Operator for deploying both the SysFlow exporter and collector as pods on OpenShift platforms. The exporter pushes SysFlow files at intervals to an S3 compliant data store, like IBM cloud object store (COS) or minio. The operator is helm based and has been tested with OpenShift 3.11 and 4.3. In the near future we hope to have a golang or ansible-based operator and have everything available in the operator catalog. For now, the operator is available on docker hub.

Prerequisites:

1. Familiarize yourself with the operator-sdk: https://github.com/operator-framework/operator-sdk/blob/master/doc/user-guide.md
2. S3 compliant object store - Currently tested with IBM's cloud object store, and minio object store (https://docs.min.io/).
   - Setup IBM Cloud Object store: https://cloud.ibm.com/docs/services/cloud-object-storage/iam?topic=cloud-object-storage-getting-started
3. Create Cloud Object Store HMAC Access ID, and Secret Key.
   - For IBM Cloud Object store: https://cloud.ibm.com/docs/services/cloud-object-storage/iam?topic=cloud-object-storage-service-credentials
4. Knowledge of OpenShift/K8s.

Clone the Repository

First, clone the repo with the following command:

```
git clone https://github.com/sysflow-telemetry/sf-deployments.git sf-deployments
cd sf-deployments/operator
```

Here are the steps in getting an OC operator deployed in an OpenShift cluster:

1. Login to your OpenShift cluster using the `oc` client tool. `oc login ...`
2. Create the sysflow project: `oc project sysflow`
3. Create a secret file for the S3 access id, and secret key. Use the following file as a template: https://github.com/sysflow-telemetry/sf-deployments/blob/master/operator/secrets.yaml and call the secret: `sfexporterchart-secrets`. Note: that access id and secret key values have to be base64 encoded in the file. You can generate the b64 encoding by doing: `echo -n <s3 access id> | base64` and copying it into the yaml file
4. Install the secret into the sysflow project: `oc create -f secrets.yaml`
5. Edit the `operator/deploy/crds/charts.helm.k8s.io_v1alpha1_sfexporterchart_cr.yaml` for your deployment:

```
apiVersion: charts.helm.k8s.io/v1alpha1
kind: SfExporterChart
metadata:
  name: sfexporterchart
spec:
  # Default values copied from <project_dir>/helm-charts/sf-exporter-chart/values.yaml

  fullnameOverride: ""
  nameOverride: sfexporter
  registry:
    secretName: ""
  resources: {}
  sfcollector:
    filter: "container.type!=host and container.name!=sfexporter
      and container.name!=sfcollector"
    interval: 300
    outDir: /mnt/data/
    repository: sysflowtelemetry/sf-collector
    tag: edge
    imagePullPolicy: Always
  sfexporter:
    interval: 30
    outDir: /mnt/data/
    repository: sysflowtelemetry/sf-exporter
    s3Bucket: sysflow-bucket
    s3Endpoint: s3.private.us-south.cloud-object-storage.appdomain.cloud
    #s3Endpoint: s3.us-south.cloud-object-storage.appdomain.cloud
    s3Location: us-south
    s3Port: 443
    s3Secure: "true"
    tag: edge
    imagePullPolicy: Always
  tmpfsSize: 500Mi
```

Most of the defaults should work in any environment. The collector is currently set to rotating files in 5 min intervals (or 300 seconds). The `/mnt/data/` is mapped to a tmpfs filesystem, and you can specify its size using the `tmpfsSize`.

For connecting to an S3 compliant data store, first take note of which port the S3 data store (`s3Port`) is configured. IBM Cloud COS listens on port 443, but certain minio installations can listen on port 9000. Also, if TLS is enabled on

the S3 datastore, ensure `s3Secure` is `true`. Ensure that the `s3Bucket` is set to the desired S3 bucket location. The `s3Endpoint` is either the domain name or IP address of the COS instance, while `s3Location` is set to a region (COS) or location (minio). Example values are above.

1. Change into the `operator` directory. There are a set of pre-defined scripts to make deployment easier.
2. Run the following commands to deploy the operator into your cluster:
    1. `./createCRD` - deploys the custom resource definition.
    2. `./deployOperator.sh` - deploys the operator proper.
    3. `./applyCR` - applies the Custom Resource.
3. The operator should now have deployed the collector and exporter.
4. Check to see that they are operational `oc get pods -n sysflow`
5. If one of the containers is not functioning properly, you can check its logs by doing: `oc logs -f <podname> -c <sfcollector or sfexporter> -n sysflow`
6. You can enter the containers via: `oc exec -it <podname> -c <sfcollector or sfexporter> -n sysflow /bin/bash`
7. To delete the operator/sysflow pod, do the following:
    1. `./cleanup.sh`
    2. `./clearCRD`

3.7 Apache License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.
```

(continues on next page)

(continued from previous page)

```
   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
```

(continues on next page)

(continued from previous page)

```
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.
```

(continues on next page)

(continued from previous page)

```
7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

3.8 Contributing

3.8.1 Contributing In General

Our project welcomes external contributions.

To contribute code or documentation, please submit a pull request to the proper github repositories.

A good way to familiarize yourself with the codebase and contribution process is to look for and tackle low-hanging fruit in the github issue trackers associated with projects. Before embarking on a more ambitious contribution, please quickly *get in touch* with us.

Note: We appreciate your effort, and want to avoid a situation where a contribution requires extensive rework (by you or by us), sits in backlog for a long time, or cannot be accepted at all!

Proposing new features

If you would like to implement a new feature, please raise an issue in the appropriate repository before sending a pull request so the feature can be discussed. This is to avoid you wasting your valuable time working on a feature that the

project developers are not interested in accepting into the code base.

Fixing bugs

If you would like to fix a bug, please raise an issue in the appropriate repository before sending a pull request so it can be tracked.

Merge approval

The project maintainers use LGTM (Looks Good To Me) in comments on the code review to indicate acceptance. A change requires LGTMs from two of the maintainers of each component affected.

For a list of the maintainers, see the MAINTAINERS.md page in the appropriate repository.

3.8.2 Legal

Each source file must include a license header for the Apache Software License 2.0. Using the SPDX format is the simplest approach. e.g.

```
/*
Copyright <holder> All Rights Reserved.

SPDX-License-Identifier: Apache-2.0
*/
```

We have tried to make it as easy as possible to make contributions. This applies to how we handle the legal aspects of contribution. We use the same approach - the Developer's Certificate of Origin 1.1 (DCO) - that the Linux® Kernel community uses to manage code contributions.

We simply ask that when submitting a patch for review, the developer must include a sign-off statement in the commit message.

Here is an example Signed-off-by line, which indicates that the submitter accepts the DCO:

```
Signed-off-by: John Doe <john.doe@example.com>
```

You can include this automatically when you commit a change to your local git repository using the following command:

```
git commit -s
```

3.8.3 Communication

Please feel free to connect with us on our Slack channel or via email. Note that the projects in this repository are not formal products. As a result, the communication channels are to the maintainers and not to a support staff.

3.8.4 Setup

The documentation is a work in progress but should provide a good overview on how to get started with the project. The Dockerfile also provides a treasure trove of information on how to build the application, dependencies, and how to test the collector.

3.8.5 Testing

This project is in its infancy and with limited resources we haven't built many testers for the projects. For the sf-collector, we do have a set of unit tests that test the coverage of most of the events of interest in `sf-collector/tests`. These tests can be run using the bats testing framework. Directions on how to install bats are in the accompanied link. To run the tests, run `bats -t tests.bat` from the tests directory. Note, that the tests also rely on python3. Before conducting a pull request, these unit tests should be run. Note, there is a version of the docker image with a `testing` tag that contains bats and the unit tests. This might be useful for testing. Also, conducting a load test and running the application under valgrind is desirable for pull requests.

3.8.6 Coding style guidelines

We follow the LLVM Coding standards where possible across the projects. There is a .clang-format file in the master repo clang-format that can be used in conjunction with ClangFormat Tool to automatically format code. For linting, we use Clang Tidy Linter. This is referenced in the sf-collector Makefile.

3.9 Code of Conduct

3.10 Contributor Covenant Code of Conduct

3.10.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

3.10.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.10.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

3.10.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

3.10.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at Slack channel or via email. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

3.10.6 Attribution

This Code of Conduct is adapted from the Qiskit project's Code of Conduct and has roots from the Contributor Covenant, version 1.4, available at version.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

T

U

V