EVE device logging redesign goals

Motivation

Logs generated from EVE services and others are currently being written to files in the file system. These logs are then read by a different EVE service called logmanager, which then batches individual logs, puts them in protobufs before exporting to cloud using APIs. With a file based mechanism and different EVE services writing to different log message files on the file system, order of logs exported to cloud cannot be guaranteed to be in the same order as they were written to files. When there is verbose logging enabled on EVE device, log files start bloating (even with log rotation in force) up to a point where they do not leave any free space for the other use cases when disk space is required. We have seen several devices getting bricked because of disk space exhaustion. It has also been observed that the logs file exported to cloud as seen by cloud are hours behind the logs seen on the device at a specific point in time.

Idea is to move away from file based logging and use a purpose built logging service like rsyslogd or fluentbit with the following goals.

- No more logging to files, unless there is a component that we cannot make to use standard syslog (eg. hypervisor logs, lisp logs etc). Even the containers launched by EVE (eg: wlan, wwan etc) should be made to use standard syslog.
- Have a disk backed queueing mechanism that saves logs from being lost in the event of unexpected power failures or reboots. This includes both main message queues and more specific action queues.
- Have mechanism to save debug logs on the device disk along with sending them to cloud. This can help engineers to access debug logs from device in the event when remote log level is not set to accept debug logs. Or should we ignore the remote log level? If we decide to persist debug logs in device, the logging infrastructure should take care of limiting the space occupied and also rotate logs without using any additional tools like linux logrotate.
- In the event of an upgrade failure, queueing mechanism should make sure to not lose the other partition (failed partition with failure messages) logs. These logs should be preserved and sent to cloud after the device comes back online.
- Have a transformer that adds the partition attribute (partition name IMGA/IMGB), eve service name and version of EVE to log messages that are exported to cloud. This helps while debugging to grep for logs specific to a particular release, partition and service.
- Ability to prioritize log queue when logs are sent cloud. Logs with different priorities should be in different queues.
- To prevent making too many API calls to cloud, logs should be exported to cloud in batches.
- The logging in /opt/zededa/bin/watchdog-report.sh should be preserved so that we get the reboot-reason. (This can be done by having the agentlog append to the reboot-reason file and avoid having to grep the log files in watchdog-report.sh)

Schema of log exported to cloud will have the following fields:

1. Time stamp of log generation
2. EVE version
3. Device image partition IMG[AB]
4. severity
5. priority
6. TAG or Service name that generated the log
7. Host name (this will typically be device UUID)
8. Any free form json specific to the source that generates the log

EVE specific and other services should directly log to syslog instead of files

There are EVE specific services (eg: zedmanager, zedrouter etc) and generic services (eg: wlan/wwan services, dnsmasq, radvd etc) that run on Edge nodes today.

EVE specific services use logrus module for message logging. Logrus module currently is made to write log messages into files on the persistent portion (/persist) of file system. Logrus provides a way to add hooks using which a TCP hook can be added for sending log messages directly to log manager service (rsyslog in this case). Syslog provides a TCP input module that can receive log messages over TCP.

Linux kit currently does not provide a syslog plugin to directly forward log messages from the services that it starts to log management service (like rsyslog). Currently all such logs generated are written to files in /var/log/ directory by linuxkit framework. But, linuxkit project has a service called memlogd that can be started during init phase. When memlogd is running, linuxkit automatically forwards all log messages to memlogd instead of files in /var/log directory. Memlogd maintains a ring buffer that stores incoming log messages. Memlogd provides a way (unixgram socket) for external services to read these log messages for process further. An EVE service can easily be written that reads logs from memlogd and forward them as they are to rsyslogd. The pre-built memlogd container from linuxkit starts memlogd with defaults that store 1K messages of 1K bytes each. We might have to inherit memlogd container from linuxkit and create our own container that starts memlogd with more space allocated for storing log messages.

There are other services like xen-commons, qemu_dm, device-steps.log that couldn't be made to send log messages to rsyslog server directly. Only option with such services is to have them log messages to files and then use "imfile" plugin of rsyslog to read them into queueing system.

Have Disk backed queueing mechanism

Rsyslog has rich support for queueing. It allows storing log message queues either in main memory or disk depending on configuration. Both main message queues (action still not decided) and action queues can be maintained on disk. Rsyslog even allows having separate main message queues per input method.

This should also take care of the case when we have base OS image upgrade failure case where edge node switches back to old image. Logs from the failure partition would have been either sent to cloud and queued in disk backed queues by rsyslogd. Rsyslogd should resume sending the queue after the edge node becomes online again.

Transformer to append EVE partition name, service name and EVE version to output json log

Rsyslog derives service name (that generated the log) from syslog message itself (using $programname in-built variable).

Rsyslog can be told about the EVE version and partition in two ways.

1. By embedding it in the log message itself. Logrus module supports "log.WithOptions" function using which we can add EVE specific meta data to the current json message sent. This will required changes in every line of code where we log something. A new go-module say "zlog" can be created that implicitly embeds partition and release information into the log messages.
   Using this method we can easily embed information into log messages generated by EVE services. But, the same cannot be done with other services like wlan/wwan, dnsmasq etc.
2. Pass partition and release version as environment variables to rsyslogd. These environment variables can be read from rsyslogd configuration and used (at a slight performance penalty). However this method has a disadvantage.
   Templates/transformers are not evaluated when logs get added to main message queue. Only after the log message is dequeued from the main message queue and processed to derive actions will rsyslog generates templated information. In the scenario where we fallback to old base image and there are logs still in main message queue (from tried/new partition), we will end up sending wrong partition/release information for log messages from the other partition.
3. Hybrid method. We use logrus to add this meta-data for EVE services and depend on environment variables for the messages from other services.

EVE log output plugin

A new EVE specific output plugin will be required that exports log messages from rsyslog on device to cloud. Job of such plugin would be:

- Protobuf format log messages
- Bundle multiple messages
- Interact with TPM (if present) on the device for signing/encrypting cloud communication
- Keep up to date with DeviceNetworkStatus generated by NIM service

There are two ways of achieving this:

1. Create a plugin written in "C" language that interacts with rsyslogd demon. This plugin written in C language would get log messages from rsyslogd and pass them on to another shared library written in golang for EVE specific processing. Interface between the C code and golang code should be kept simple with primitive data types. This approach makes is very easy for exerting back pressure on the message queue when there are network or other failures due to which log messages cannot be delivered to cloud.
2. If the above approach does not work or has issues, we can always implement the EVE specific functionality as a separate process and have rsyslogd forward messages to our new service using omtcp module. With this there is a problem that the message in transit (from rsyslogd to EVE forwarder service) will always be lost when connectivity to cloud fails.

** Question for Roman - How do we build the plugin in our build environment?

Avoid making too many API calls to cloud for sending log messages

Rsyslog supports batching of log messages before sending to output plugin. Rsyslogd starts a transaction, sends a a bunch of logs and then ends transaction. Plugin has the option to selectively acknowledge a subset of log messages or reject all. Rsyslogd does not mark these log messages as completed until EVE plugin acknowledges/accepts these log messages. This mechanism can be used for putting back pressure to rsyslogd in the event of network or other failures.

Envisioned list of log sources on an EVE edge-node

1. EVE specific agents. This include EVE agents like zedagent, zedrouter, domainmgr, verifier, zedmanager, downloader, identitymgr, vaultmgr, baseosmgr, nim, nodeagent, ledmanager, wstunnelclient, lisp-ztr and python lisp control plane.
2. External tools used by EVE. This include dnsmasq, radvd, watchdog, dhcpcd
3. EVE specific scripts and short running executables. eg. client, diag, device-steps.sh
4. Other containers running in EVE control domain (dom0). eg. wwan, wlan, guacd, sshd, vtpm (vtpm_server) and ntpd containers.
5. Hypervisor and associated tools. eg. xen, xenstored, xenconsoled, xl, qemu.
6. Kernel logs.

These log sources can broadly be grouped into these five categories based on how their logs are output (log destinations):

1. Sources that send logs only to files. Such sources include hypervisor, qemu, lisp control plane, xen-tools.
   These logs will need special handling. We can start with having such logs be dumped to /var or /persist and use imfile module of rsyslogd to pick up from there.
   We should later invent a mechanism like LD_PRELOAD or named pipes to makes these sources send logs to rsyslogd without using files.
   There have been mixed opinions from team. Having such services keep logging to files and then make imfile module of rsyslogd scrape logs from file is an option. This will mean that
   we might need aggressive log management (archiving old log files and aggressively deleting the oldest archives).
2. Sources that are flexible and can be made to change their log destination easily. Such sources include EVE agents, short lived EVE executables and scripts.
   EVE agents for examples can be changed (since they use logrus for their logging needs) to send their logs directly to syslog (/dev/log) or stdout and then re-direct to rsyslogd using logger tool.
   After discussion with team the most preferred way for EVE services/executables is to have an env variable the presence/absence of which will make EVE services log to syslog or stdout directly.
3. Sources that send logs to both syslog and files. Eg. dnsmasq, dhcpcd, radvd, watchdog etc.
   No special handling will be required in this case.
4. Sources that run inside containers and the logs of which are collected by memlogd. Eg. wlan, wwan, sshd, guacd, ntpd, vtpm etc.
   Linuxkit has a module called memlogd that collects container logs in a circular buffer. Linuxkit's memlogd module is shipped with logread tool that

can read logs from memlogd and output to it's stdout.
Output of logread can be piped into logger and subsequently sent to rsyslogd.

5. Kernel logs (easily collected by rsyslogd). Rsyslogd has a module called imklog that can read kernel logs into rsyslogd.
   ** Question: Can rsyslogd user imklog to read xen logs directly?