Debugging APIs

We currently have the device send log information to the controller using the log API, and this can be useful when debugging issues in EVE.

However, in some cases it is useful to also be able to inspect the current state. That state could be the state maintained by the EVE microservices (e.g., the AppInstanceStatus maintained by zedmanager), or it could be external state such as the iptables or ps process output.

This proposal specifies how a well-defined set of such information can be retrieved by the controller.

Background and motivation

We currently deliver the logs from the EVE microservices to the controller, plus specific information relating to the device and instance status and metrics. However, two issues makes it harder to use those logs than those on the device, the first being that they are consolidated from all the agents, and the second being that the logs are for the lifetime of the device (split in IMGA and IMGB logs) and in most cases one cares about what happened after the last reboot.

In addition, the current state of the device is easier to determine by examining /var/run on the device, and looking at things like the output of ps or xl list.

Finally, there are implementation internal aspects (such as iptables -L, ip rule show, ip route show) which are useful when debugging issues.

Proposal

We already have the logging API as a flexible and scaleable way to deliver information from the device to the controller, with the appropriate retry /retransmission logic in EVE. Its only constraint is that a single log item must be smaller than the maximum size configured in the web server running on the controller.

We also have a flexible way to extend the configuration using the ConfigItem message in the configuration; a string key plus a string value, which is used for timer and policy settings.

Last but not least we have a way to send commands such as the RebootCmd using eventual consistency by having a counter to ensure that a command is executed at least once.

Combining those we can add support for additional debug commands by defining a ConfigItem key string for each, where the value is a number. When the device receives such a ConfigItem it checks if the number is different than what it last processed for that particular key, and if it is the device performs the operation and the output is sent to the log API.

Initial set of keys/commands

Command	Reported information	Potential use
ps	ps output	Look for hung processes
du	du -a /persist	Track down disk usage
du.<subdir>	du -a /persist.<subdir>	E.g., du.log, du.IMGA
state	All of /var/run content	Snapshot for all agents and object
state.<agent>	/var/run/<agent>	Snapshot for one agent
state.<agent>.<type>	/var/run/<agent>/<type>	For agent and type
state.obj.<key>	/var/run/*/*/<key>	E.g., look for an instance UUID
config	/config except any *.key.pem	Looking for stale files
persist.<subdir>	ls /persist/<subdir>	Looking for stale files or missing certs
lspci	Alpine lspci output	Check if pci controllers match model
lsusb	Alpine lsusb output	Check if any USB devices connected
iptables	Iptables -t filter; iptables -t raw; iptables -t nat, all with -L -nv	Check if iptables are wrong + counters
route	ip route show	
route.X	ip route show table X	
rule	ip rule show	

Considerations for adding future commands

For security reasons any command should be of a fixed function; no command should ever allow arbitrary execution of e.g., shell commands. Furthermore, when defining new commands one needs to take care to not expose any secret information from the device, such as the content of running edge container objects, or credentials for datastore access.

Currently none of the defined commands alter the state of the device, and if there is a desire to alter the state (e.g., purge certain directories to recover from low on disk space) it would make sense to explore alternative approaches than this basic fire-and-forget approach.

Implementation notes

The device will retain counter Y value for command string X, in similar ways as it retains a rebootCount and uuidtonum persistently across reboots.

This could be in /persist/status/zedagent/KeyToNum/X.json

When zedagent receives config items from the controller it will compare the counter Y with what is recorded, and if it is different than it will send the requested output to the log API. It makes sense for the log output to include the command string and counter value.