Debug Docker Images – An in-depth view

Problem space:

1) Today to do live debugging or dead-core-file debugging, one need to follow the steps below.
    a. Get a physical/virtual DUT.

    b. Download & install the required sonic version.
        i. This means if you are looking at multiple cores from different versions, you would need multiple DUTs as one per version

    c. <**Harder step**> Understand the daemon you are debugging in-depth so as to know the list of debug packages you would need.
        i. For an example, if you need to debug syncd from SONiC.20180330.108, in addition to the obvious, syncd's daemon's dbg package(syncd-dbg_1.0.0_amd64.deb), you also need the following
            - libsaibcm-dbg_3.1.3.4-19_amd64.deb
            - libsaimetadata-dbg_1.0.0_amd64.deb
            - libsairedis-dbg_1.0.0_amd64.deb

        ii. Getting the right version of the symbol is not often straight forward,
            - For an example to find libsaibcm version to use, you need to look at platform/Broadcom/sai.mk file
            - Plus you need to have the complete tree of 'debs' preserved

    d. Get these DBG packages to your DUT and install these into your docker.

    e. Install gdb & other tools you may need to facilitate your debugging
        i. To make it easy, you would need internet access, so you can do "apt install …"
        ii. If not, it is tougher to get them.

    f. No source code access
        i. With all the above, you get symbols and you can see meaningful info
        ii. But still you don't have the luxury of debugging with live source code mapping.

Result:

1) Devs tend to avoid debugging cores, as they don't have time for this overhead.
2) This leads to the state of turning ***easy*** debugging effort to look monumental and it gets deemed as ***only experts can do***.
3) We end up not finding bugs in time, and tend towards mitigating/work around.
4) These ***monotonous***, and time consuming efforts spent in setup are ***duplicated*/*repeated*** for each debugging session.

Goals:

1) Make debugging as easy as it should be

2) Take monotonous steps off of the hands of devs.
3) Use dev's time for *__only__* the area where they are good at – that is look at the crashing lines of code
4) Make basic debugging *__automatable__*
5) Enable in-depth debugging *__automatable__* using dev provided gdb-python scripts committed along with source code.

How do we do:

1) Each target, declares its list of debug packages and the custom tools that might be of use to devs for debugging, in addition to including those declared by its base package.
   We extend this service only to stretch dockers.
   If you convert your docker to stretch, you may use the following example, to set it up.

   Example: Following are the updates to docker-orchagent.mk

'DOCKER_ORCHAGENT_STEM = docker-orchagent'	Declare the base name as stem, so it enables suffixing with "-dbg"
'DOCKER_ORCHAGENT = $(DOCKER_ORCHAGENT_STEM).gz'	Declare docker name using stem
'DOCKER_ORCHAGENT_DBG = $(DOCKER_ORCHAGENT_STEM)-$(DBG_IMAGE_MARK).gz'	Declare DBG docker name with DBG_IMAGE_MARK which is declared as "dbg" in slave.mk
'$(DOCKER_ORCHAGENT)_DEPENDS += $(SWSS) $(REDIS_TOOLS)'	*<As of now>*
'$(DOCKER_ORCHAGENT)_DBG_DEPENDS = $($(DOCKER_CONFIG_ENGINE_STRETCH)_DBG_DEPENDS)'	Acquire all the DBG packages required by your immediate base package, which is in this case docker-config-engine-stretch
'$(DOCKER_ORCHAGENT)_DBG_DEPENDS += $(SWSS_DBG) \ $(LIBSWSSCOMMON_DBG) \ $(LIBSAIREDIS_DBG)'	Add your own DBG packages
'$(DOCKER_ORCHAGENT)_DBG_IMAGE_PACKAGES = $($(DOCKER_CONFIG_ENGINE_STRETCH)_DBG_IMAGE_PACKAGES)'	Acquire tools required to be installed in the docker using 'apt-get install'. The orchagent could add some custom tools, it might need. [If name _DBG_IMAGE_PACKAGES is not *__clear__*, we could switch to something more explicit, like _DBG_TOOLS ?)
'$(DOCKER_ORCHAGENT)_PATH = $(DOCKERS_PATH)/$(DOCKER_ORCHAGENT_STEM) $(DOCKER_ORCHAGENT)_LOAD_DOCKERS += $(DOCKER_CONFIG_ENGINE_STRETCH)	*<As of now>*

SONIC_DOCKER_IMAGES += $(DOCKER_ORCHAGENT) SONIC_STRETCH_DOCKERS += $(DOCKER_ORCHAGENT) SONIC_INSTALL_DOCKER_IMAGES += $(DOCKER_ORCHAGENT)'	
'SONIC_DOCKER_DBG_IMAGES += $(DOCKER_ORCHAGENT_DBG) SONIC_STRETCH_DBG_DOCKERS += $(DOCKER_ORCHAGENT_DBG) SONIC_INSTALL_DOCKER_DBG_IMAGES += $(DOCKER_ORCHAGENT_DBG)'	Add self to DOCKER_DBG_IMAGES – which will be built If you are stretch based, add self to SONIC_STRETCH_DBG_DOCKERS, to distinguish, so as to facilitate "make stretch" which only does stretch based. Add self to SONIC_INSTALL_DOCKER_DBG_IMAGES, to facilitate self into final debug image (.bin, .swi, .raw, …)
'$(DOCKER_ORCHAGENT)_CONTAINER_NAME = swss $(DOCKER_ORCHAGENT)_RUN_OPT += --net=host --privileged -t $(DOCKER_ORCHAGENT)_RUN_OPT += -v /etc/network/interfaces:/etc/network/interfaces:ro $(DOCKER_ORCHAGENT)_RUN_OPT += -v /etc/network/interfaces.d/:/etc/network/interfaces.d/:ro $(DOCKER_ORCHAGENT)_RUN_OPT += -v /host/machine.conf:/host/machine.conf:ro $(DOCKER_ORCHAGENT)_RUN_OPT += -v /etc/sonic:/etc/sonic:ro $(DOCKER_ORCHAGENT)_RUN_OPT += -v /var/log/swss:/var/log/swss:rw $(DOCKER_ORCHAGENT)_BASE_IMAGE_FILES += swssloglevel:/usr/bin/swssloglevel $(DOCKER_ORCHAGENT)_FILES += $(ARP_UPDATE_SCRIPT) $(SUPERVISOR_PROC_EXIT_LISTENER_SCRIPT)'	*<As of now>*

2) For each docker included in SONIC_DOCKER_DBG_IMAGES, a debug docker image (docker-<name>-dbg.gz) is built

3) To debug a core,
    a. Pick a system that can host/run docker (May be your dev box/VM)
    b. Download the corresponding debug docker image from build repository of right version and load it.
    c. Download & unzip core file

    d. [*optional*] clone & switch to the right <branch/tag> matching the version including submodules
        i. Make init does this transparently for most modules
    e. Run the docker image with core & optionally source path mapped with entrypoint forced to /bin/bash.
        i. Docker run -v <core file path>:/core [-v <source-file-path>:/src] --entrypoint=/bin/bash> <image name/id>
    f. In docker, run gdb and ***expect*** all debug symbols in place.

4) To live debug
    a. Download the appropriate debug version of docker image.
    b. Load the current image and run
    <NOTE: I am yet to test, this to be able to give precise steps. But doable. If anyone can volunteer, please help>

5) <*Work in Progress*> In addition, we build a final debug image too (e.g. sonic-broadcom-dbg.bin) too.
    a. Same as original image, but where available it uses debug docker image instead of regular one.
    b. This can be installed in any DUT and can run normally, except that there can be performance downgrade due to added debug symbols.

Gain:

1) To debug all you need is the debug docker image
2) You are good to debug in any environment with no constraints
3) No additional overhead to ensure debug symbols or tools.
4) Concurrently debugging cores from different versions is easy.

Cost:

As ***nothing is for free***, there is indeed a cost associated, which is build-time cost.

Pre-condition/Unassociated cost:

The cost of building debug packages for each target is ***required*** irrespective of whether we build debug docker image or not. This is because, w/o it, it totally strips off of our ability to debug, with any overhead taken, unless we checkout the right version of source & build it ourselves, which is a lost cause.

Current cost per Debug docker image:

1) Couple of “j2” runs, where each run creates a text file using Jinja2 template
2) Build Debian file system (build_debian.sh)
3) Build docker

To improve/cost-cut-down:

1) Share j2 runs & Debian build with regular image

2) Only added cost would be “docker build” command
    a. This would build off of pre-built docker image
    b. With only add-ons
        i. Install debug packages
        ii. Install debug tools