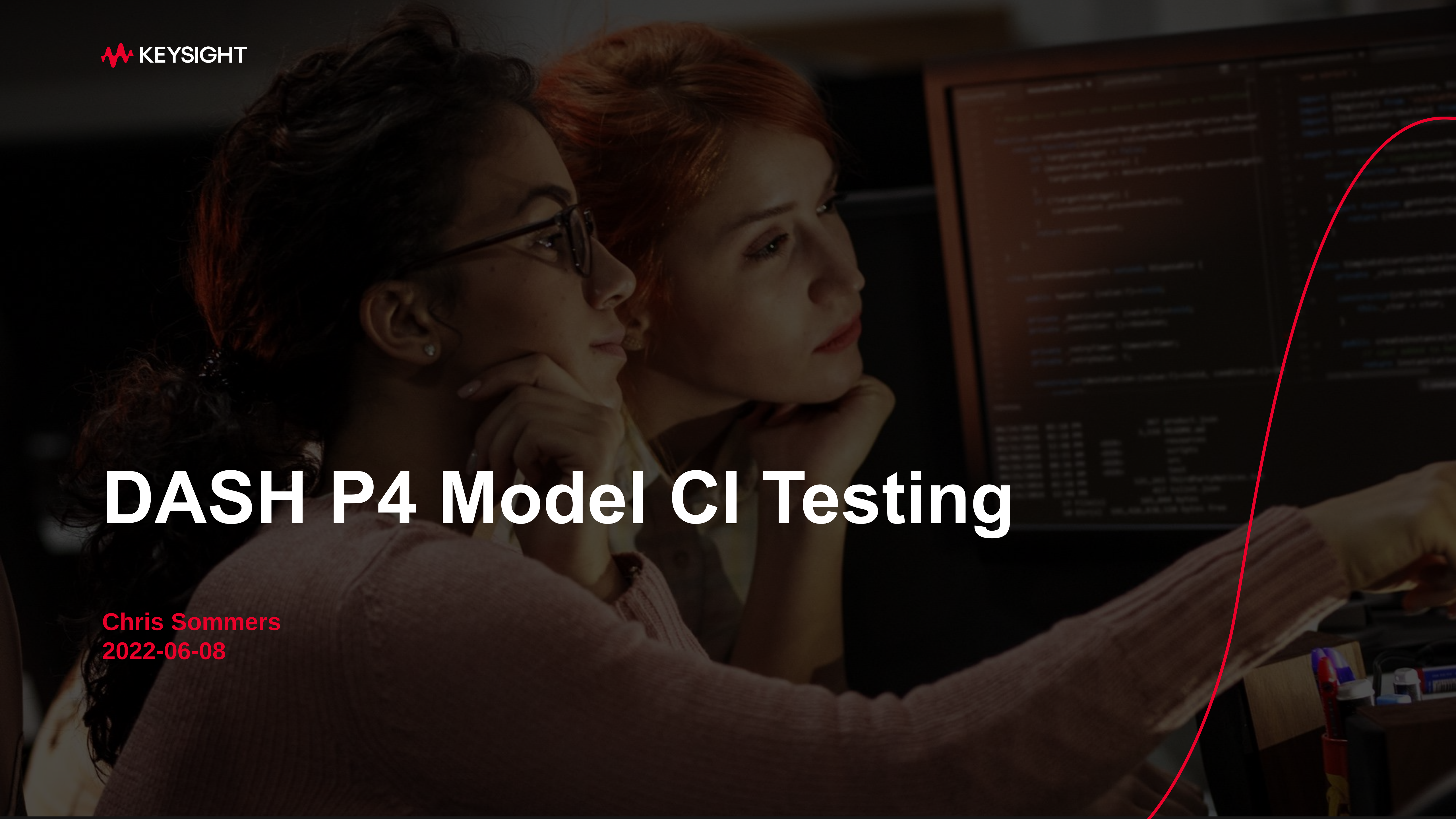


KEYSIGHT

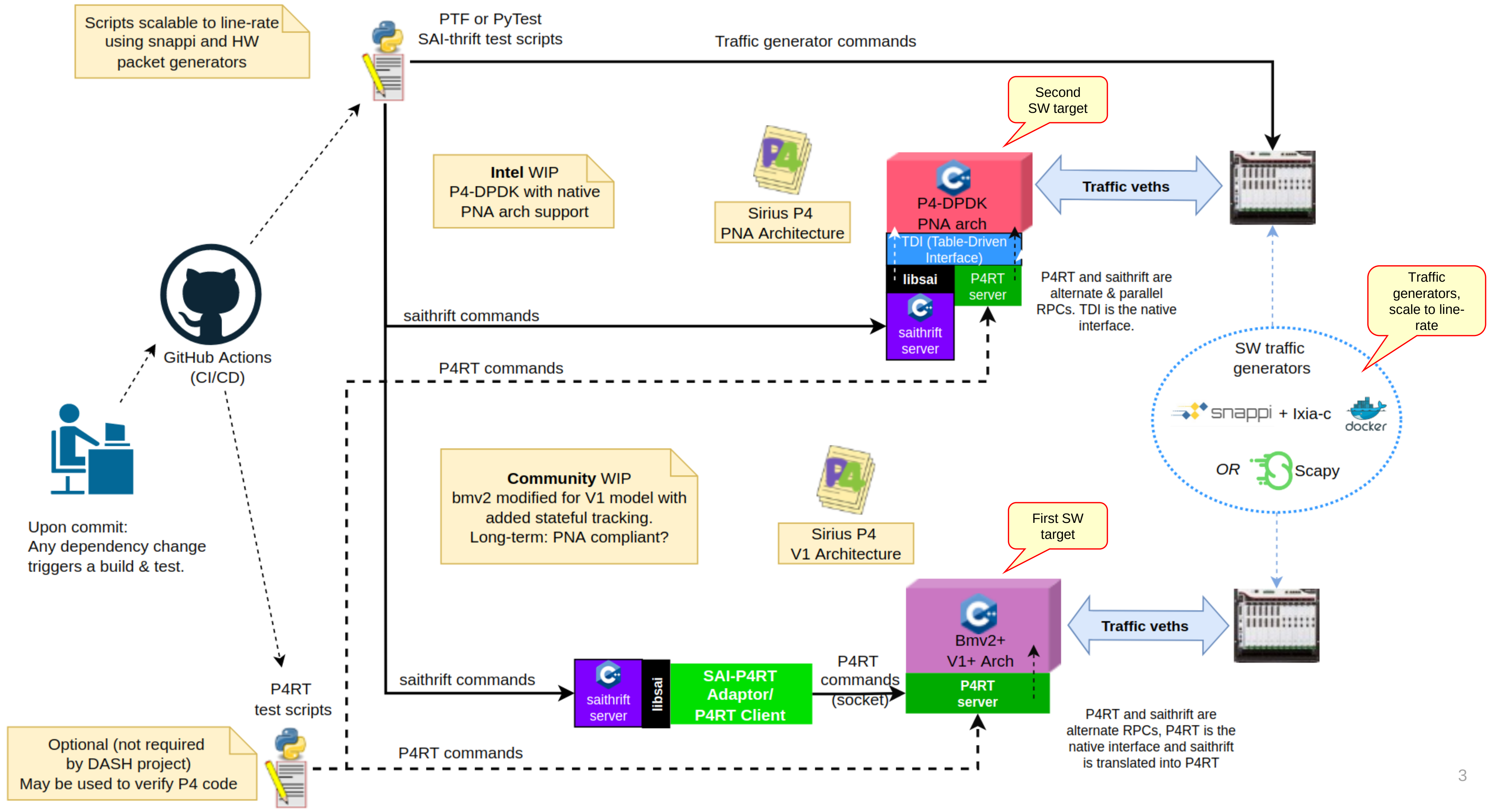
DASH P4 Model CI Testing

Chris Sommers
2022-06-08

DASH P4 Model CI Testing - Outline

- Goals/Objectives
- Progress to date
- Dependencies
- Next Steps

DASH Testing – P4 Model, multiple SW targets



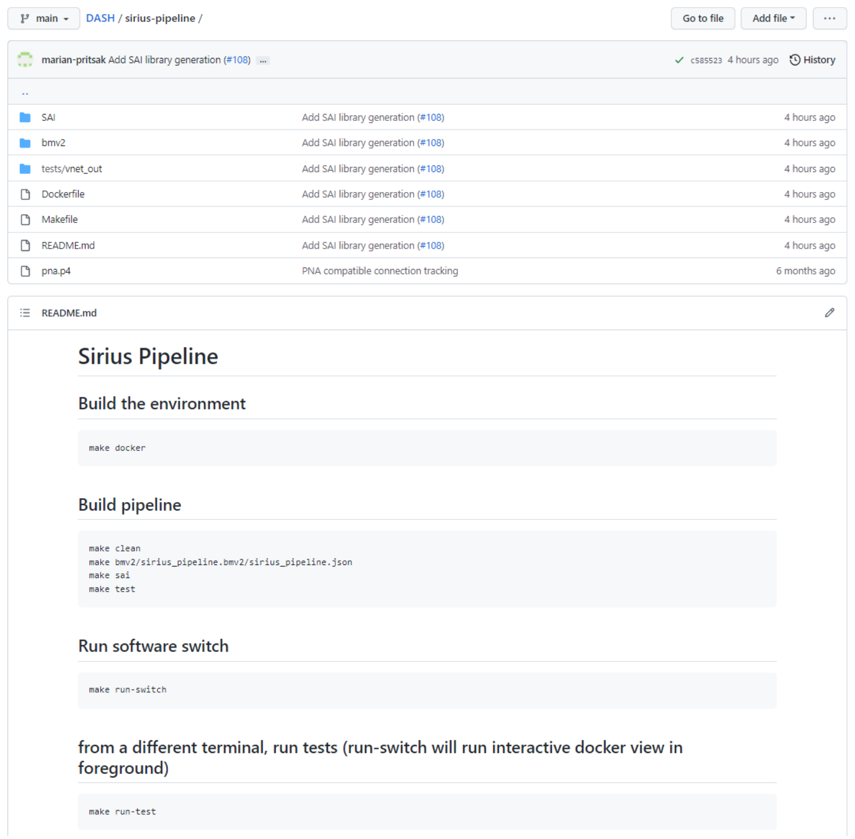
Scripts scalable to line-rate using snappi and HW packet generators
PTF or PyTest SAI-thrift test scripts
Traffic generator commands
Second SW target
Intel WIP P4-DPDK with native PNA arch support
Sirius P4 PNA Architecture
P4-DPDK PNA arch
TDI (Table-Driven Interface)
libsai
P4RT server
saithrift server
Traffic veths
P4RT and saithrift are alternate & parallel RPCs. TDI is the native interface.
Traffic generators, scale to line-rate
SW traffic generators
snappi + Ixia-c
docker
OR
Scapy
GitHub Actions (CI/CD)
saithrift commands
P4RT commands
Upon commit: Any dependency change triggers a build & test.
Community WIP bmv2 modified for V1 model with added stateful tracking. Long-term: PNA compliant?
Sirius P4 V1 Architecture
First SW target
Bmv2+ V1+ Arch
Traffic veths
P4RT test scripts
saithrift commands
saithrift server
libsai
SAI-P4RT Adaptor/ P4RT Client
P4RT commands (socket)
P4RT server
P4RT and saithrift are alternate RPCs, P4RT is the native interface and saithrift is translated into P4RT
Optional (not required by DASH project) May be used to verify P4 code
P4RT commands

DASH P4 Model CI Testing - Goals

- Produce a Framework which can perform SW regression testing in the cloud
- Use “Git Actions” triggered by a commit to the repo (e.g. to a dev branch)
- The following to be tested, directly or indirectly:
  - Sirius pipeline P4 code – compiles correctly
  - P4info -> SAI header generation
  - Bmv2 switch and P4Runtime server build & execute
  - Sai library -> P4runtime client
  - Trivial sai table accessors in c++
  - Sai-thrift server integration
  - Sai-thrift configuration of P4 D.U.T (API)
  - Traffic generator spin-up
  - Traffic tests using bmv2 veth ports (Dataplane)
  - Longer-term – test sai-redis, gNMI
  - Longer-term – line-rate testing on real HW

Working as of today (2022-06-07)
Manual using make commands

TODO!



DASH / sirius-pipeline /

marian-pritsak Add SAI library generation (#108) ... c585523 4 hours ago

..		
SAI	Add SAI library generation (#108)	4 hours ago
bmv2	Add SAI library generation (#108)	4 hours ago
tests/vnet_out	Add SAI library generation (#108)	4 hours ago
Dockerfile	Add SAI library generation (#108)	4 hours ago
Makefile	Add SAI library generation (#108)	4 hours ago
README.md	Add SAI library generation (#108)	4 hours ago
pna.p4	PNA compatible connection tracking	6 months ago

README.md

Sirius Pipeline

Build the environment

```
make docker
```

Build pipeline

```
make clean
make bmv2/sirius_pipeline.bmv2/sirius_pipeline.json
make sai
make test
```

Run software switch

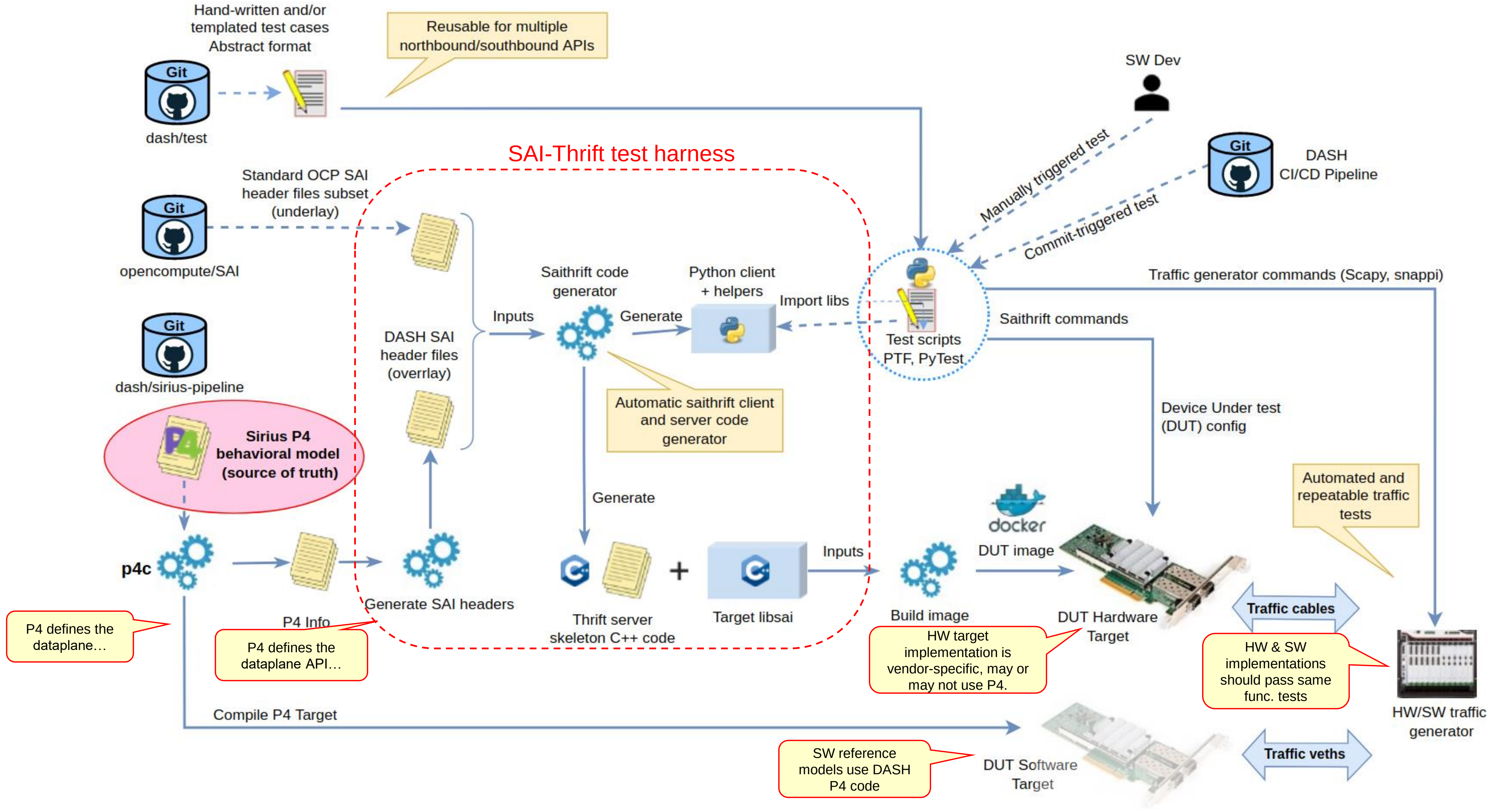
```
make run-switch
```

from a different terminal, run tests (run-switch will run interactive docker view in foreground)

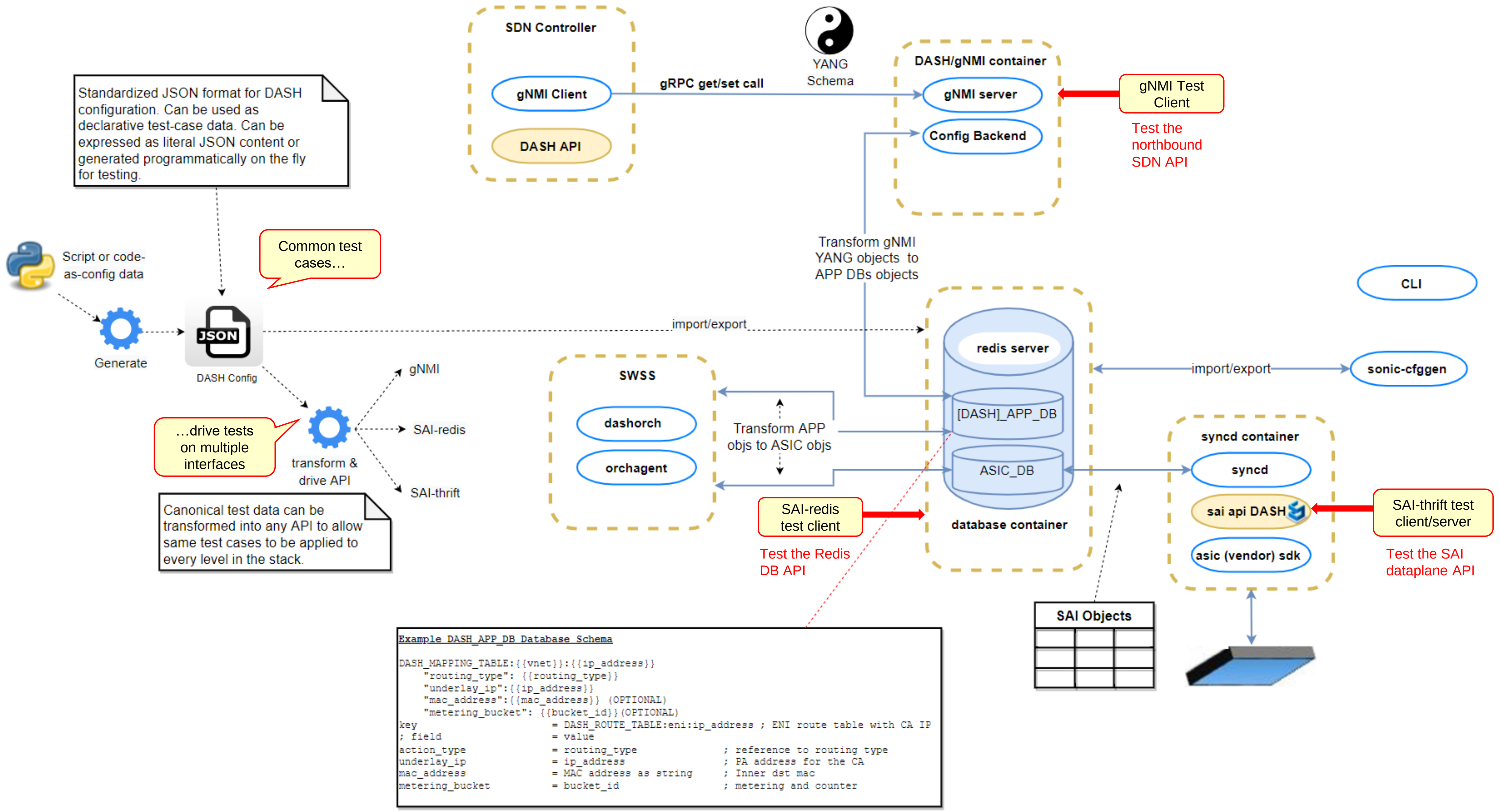
```
make run-test
```

KEYSIGHT

DASH Testing – Workflows & auto-generated artifacts


Hand-written and/or templated test cases Abstract format
Reusable for multiple northbound/southbound APIs
Git
dash/test
SW Dev
SAI-Thrift test harness
Manually triggered test
Git
DASH CI/CD Pipeline
Standard OCP SAI header files subset (underlay)
Commit-triggered test
Git
opencompute/SAI
Saithrift code generator
Python client + helpers
Import libs
Traffic generator commands (Scapy, snappi)
Inputs
Generate
Saithrift commands
Git
DASH SAI header files (overlay)
Test scripts PTF, PyTest
dash/sirius-pipeline
Automatic saithrift client and server code generator
Device Under test (DUT) config
Sirius P4 behavioral model (source of truth)
Generate
Automated and repeatable traffic tests
docker
DUT image
Inputs
p4c
Generate SAI headers
Build image
DUT Hardware Target
Traffic cables
P4 Info
Thrift server skeleton C++ code
Target libsai
P4 defines the dataplane…
P4 defines the dataplane API…
HW target implementation is vendor-specific, may or may not use P4.
HW & SW implementations should pass same func. tests
HW/SW traffic generator
Compile P4 Target
SW reference models use DASH P4 code
DUT Software Target
Traffic veths

DASH Testing – API/Schema layers, common test cases



Standardized JSON format for DASH configuration. Can be used as declarative test-case data. Can be expressed as literal JSON content or generated programmatically on the fly for testing.

Canonical test data can be transformed into any API to allow same test cases to be applied to every level in the stack.

```
Example DASH_APP_DB Database Schema

DASH_MAPPING_TABLE:{{vnet}}:{{ip_address}}
    "routing_type": {{routing_type}}
    "underlay_ip":{{ip_address}}
    "mac_address":{{mac_address}} (OPTIONAL)
    "metering_bucket": {{bucket_id}}(OPTIONAL)
key                         = DASH_ROUTE_TABLE:eni:ip_address ; ENI route table with CA IP
; field                     = value
action_type                 = routing_type              ; reference to routing type
underlay_ip                 = ip_address                ; PA address for the CA
mac_address                 = MAC address as string     ; Inner dst mac
metering_bucket             = bucket_id                 ; metering and counter
```

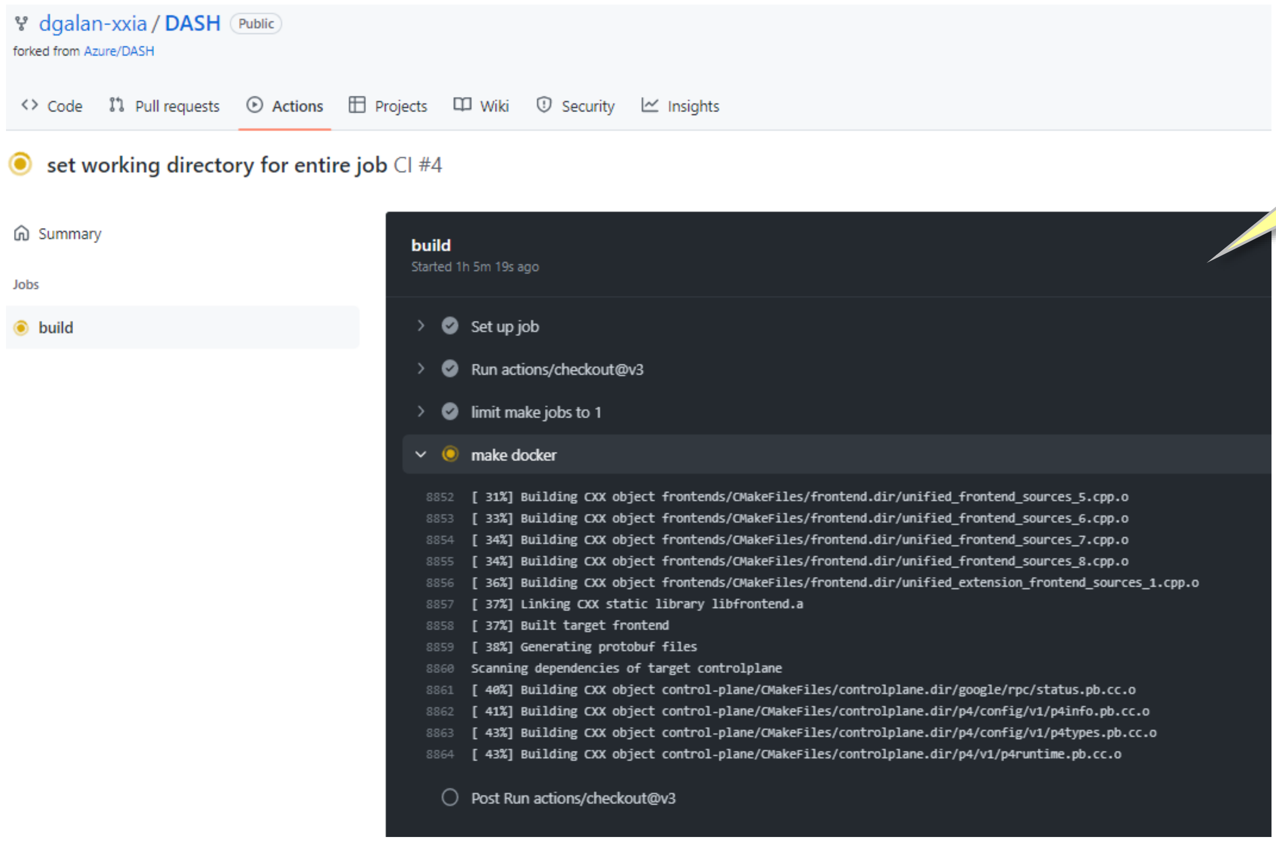
What happens when Git Action is triggered?

- Test Runner is allocated in Azure cloud – free for public projects
- Build (or retrieve a pre-built) docker image containing tool chain: gRPC, thrift, p4c, etc.
- Using the docker env, compile P4 model, produce all artifacts (bmv2 executable, SAI headers, SAI-P4Runtime adaptor, sai-thrift server.
- Launch P4 switch + Sai-thrift server
- Launch Docker containers with ixia-c traffic generator (free version) – supports snappi/OTG*
- Run Pytests to configure dataplane, send traffic, analyze results
- Pass/fail report, Github status badge
- Shut down

* See references at end of deck

Progress to date – just getting started…

- Analyzing framework design, resources, schedule, dependencies.
- Starting to build dockers triggered by Github actions. Docker build takes a long time, we need to store somewhere and retrieve as needed.



dgalan-xxia / DASH Public
forked from Azure/DASH

set working directory for entire job CI #4

build
Started 1h 5m 19s ago

- Set up job
- Run actions/checkout@v3
- limit make jobs to 1
- make docker

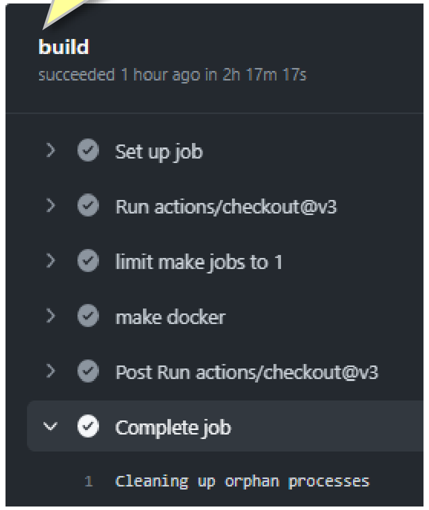
```
[ 31%] Building CXX object frontends/CMakeFiles/frontend.dir/unified_frontend_sources_5.cpp.o
[ 33%] Building CXX object frontends/CMakeFiles/frontend.dir/unified_frontend_sources_6.cpp.o
[ 34%] Building CXX object frontends/CMakeFiles/frontend.dir/unified_frontend_sources_7.cpp.o
[ 34%] Building CXX object frontends/CMakeFiles/frontend.dir/unified_frontend_sources_8.cpp.o
[ 36%] Building CXX object frontends/CMakeFiles/frontend.dir/unified_extension_frontend_sources_1.cpp.o
[ 37%] Linking CXX static library libfrontend.a
[ 37%] Built target frontend
[ 38%] Generating protobuf files
Scanning dependencies of target controlplane
[ 40%] Building CXX object control-plane/CMakeFiles/controlplane.dir/google/rpc/status.pb.cc.o
[ 41%] Building CXX object control-plane/CMakeFiles/controlplane.dir/p4/config/v1/p4info.pb.cc.o
[ 43%] Building CXX object control-plane/CMakeFiles/controlplane.dir/p4/config/v1/p4types.pb.cc.o
[ 43%] Building CXX object control-plane/CMakeFiles/controlplane.dir/p4/v1/p4runtime.pb.cc.o
```

- Post Run actions/checkout@v3

After one hour, still building docker …zzzzzzz

Only took 1h:47m but failed making bmv2

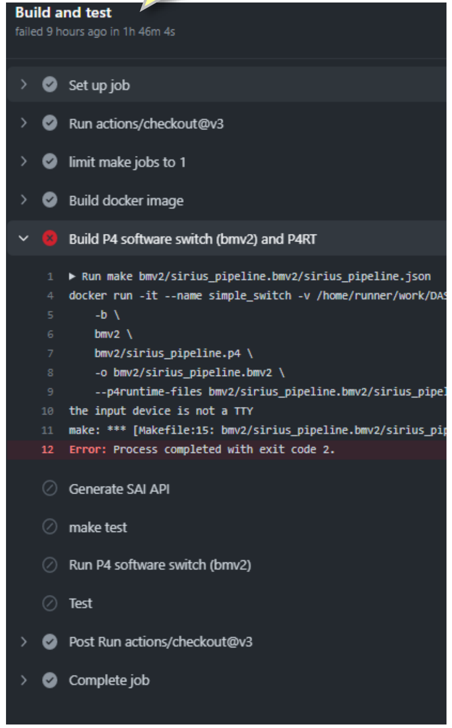
Docker build 2h 17 min



build
succeeded 1 hour ago in 2h 17m 17s

- Set up job
- Run actions/checkout@v3
- limit make jobs to 1
- make docker
- Post Run actions/checkout@v3
- Complete job

```
Cleaning up orphan processes
```



Build and test
failed 9 hours ago in 1h 46m 4s

- Set up job
- Run actions/checkout@v3
- limit make jobs to 1
- Build docker image
- Build P4 software switch (bmv2) and P4RT

```
▶ Run make bmv2/sirius_pipeline.bmv2/sirius_pipeline.json
docker run -it --name simple_switch -v /home/runner/work/DAS
    -b \
    bmv2 \
    bmv2/sirius_pipeline.p4 \
    -o bmv2/sirius_pipeline.bmv2 \
    --p4runtime-files bmv2/sirius_pipeline.bmv2/sirius_pipel
the input device is not a TTY
make: *** [Makefile:15: bmv2/sirius_pipeline.bmv2/sirius_pip
Error: Process completed with exit code 2.
```

- Generate SAI API
- make test
- Run P4 software switch (bmv2)
- Test
- Post Run actions/checkout@v3
- Complete job

KEYSIGHT

Dependencies

- sai-thrift server interface to bmv2
  - DASH enhancements to sai-thrift merged (Intel) – note, DASH can use a dev branch for now
  - Need to integrate into docker builds so a complete sai-thrift capable bmv2 is built (Need volunteer)
- Choose some simple exemplary test cases (community)
- Semi-stable P4 model which can pass some defined traffic tests, preferably stateful + stateless (community)
- Stable SAI interfaces to DASH (should derive from stable P4 code)
- Declarative device config schema, e.g. JSON format (MSFT)
  - Will be used to drive device API operations through adaptors (initially sai-thrift; later sai-r4dis, gNMI)
  - Preferably we can import/export to/from redis using these same files.
- Docker image repository to avoid rebuilding stable tools (MSFT, Keysight)
- More powerful Github runners (does sonic-buildimage use them)? (MSFT)

Next Steps

- Get basic CI working (build the artifacts, run existing trivial sai test) - Keysight
- Sai-thrift enhancements for DASH – Intel
- Sai-thrift server integration to Sirius pipeline libsai – need volunteer
- Test harness framework – Keysight (analyzing requirements)
- Choose test cases - community
- Stabilize P4 model to some known level – DASH Behavioral Model WG
- Define JSON config schema – MSFT, community
- Identify a suitable docker repo to store tools image – MSFT, Keysight
- Conduct regular Test WG meetings?

- Questions/Feedback? Thank you!

References

- Goodbye Scapy hello snappi – YouTube (https://www.youtube.com/watch?v=Db7Cx1hngVY)
- Open Traffic Generator snappi Ixia-c – YouTube (https://www.youtube.com/watch?v=3p72YnLFZVQ)
- https://github.com/open-traffic-generator • https://github.com/open-traffic-generator/snappi
- https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners

- https://github.com/opencomputeproject/SAI/tree/master/test/saithriftv2

P4 Workshop 2022 Talk - Chris Sommers (Keysight) and Reshma Sudarshan (Intel):

- https://www.youtube.com/watch?v=mT7-t_aDozM – video
- https://opennetworking.org/wp-content/uploads/2022/05/Reshma-Sudarshan-Chris-Sommers-Final-Slide-Deck.pdf - Slides