ONAP: OpenDaylight Decoupling

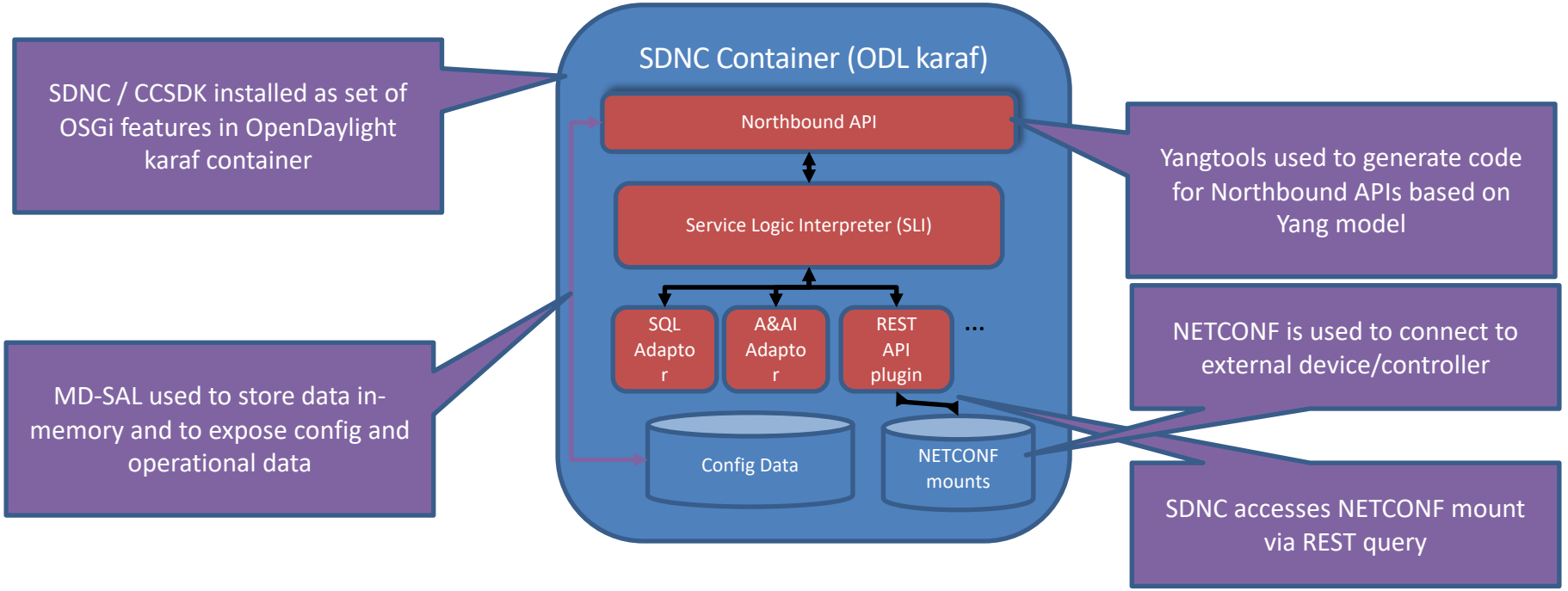
Dan Timoney - AT&T

@djtimoney

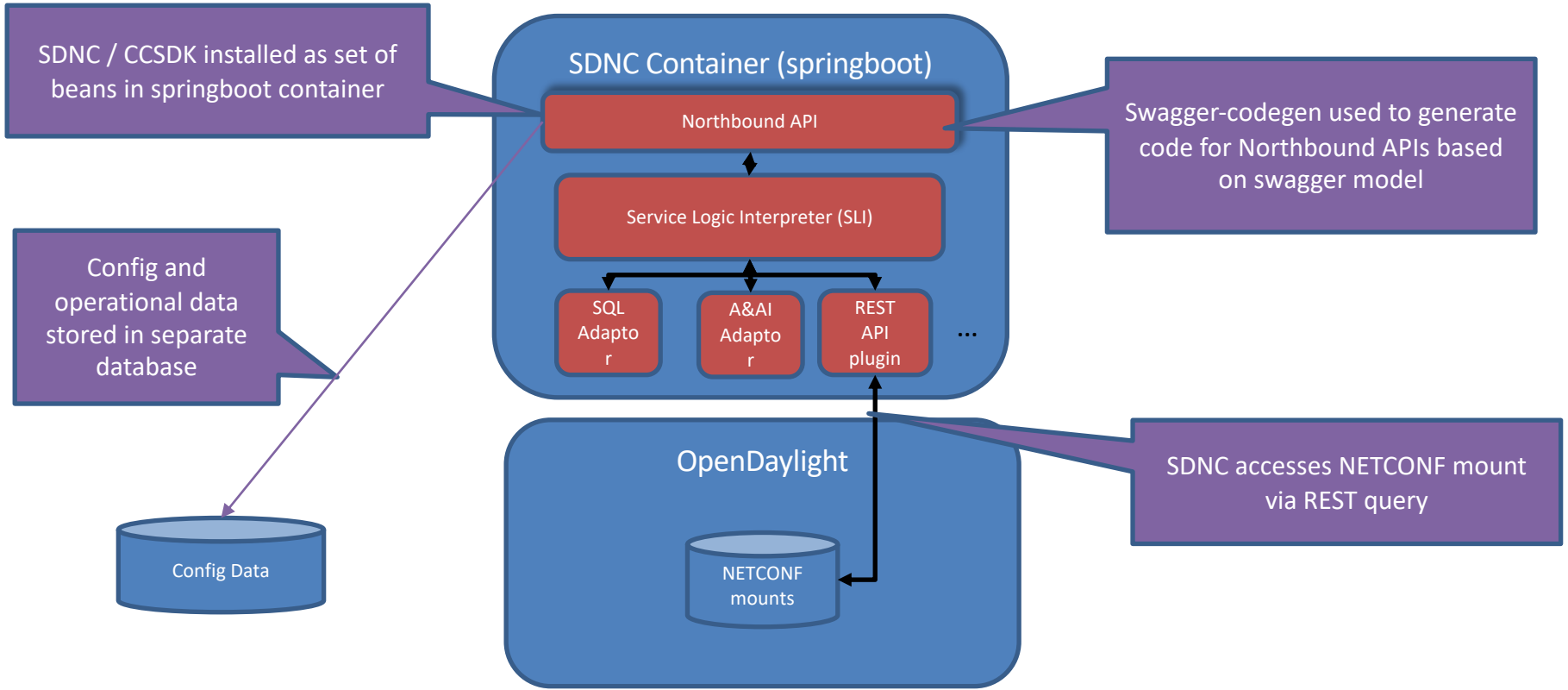
Problem Statement

- A given release CCSDK/SDNC code is compiled for a specific release of OpenDaylight:
  - ONAP Honolulu release is currently based on OpenDaylight Aluminum SR3
- What if you need to run SDNC with a different version of OpenDaylight (for example, to address a bug fix)?
  - Make a local copy of CCSDK parent poms (ccsdk/parent repo)
  - Update versions of OpenDaylight and third-party libraries pre-installed in OpenDaylight distribution
  - Compile CCSDK and SDNC repos against updated local version of parent poms
    - Most OpenDaylight major releases have some breaking changes that require code changes. So, some code changes could be needed.
  - Create new dockers using locally compiled version
- Our goal is to support a much simpler process:
  - Run SDNC in a separate container from OpenDaylight, and use the versions you need.
  - OR, if you can't (e.g. due to performance concerns), create your own dockers, based on a docker container containing the OpenDaylight version you need.

Current model : SDNC within OpenDaylight



Decoupled model : SDNC separated from OpenDaylight



Implementing Decoupled Model – Phase 1

- Work spans multiple ONAP releases:
  - Guilin:
    - Refactored code to isolate dependencies on OpenDaylight
    - Updated CCSDK components to allow SLI to consume adaptors and plugins as beans when running outside OSGi
    - Implemented a simple springboot container implementing SLI-API (healthcheck).

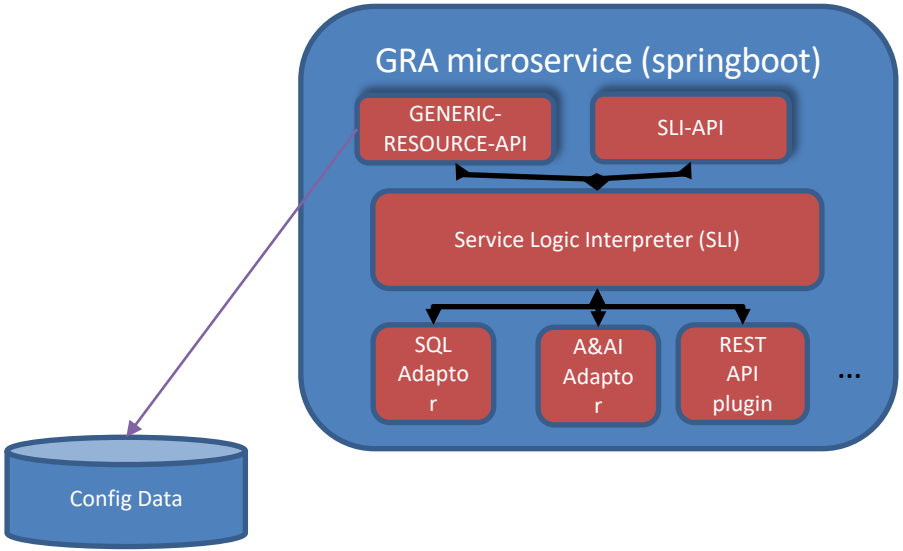
Implementing Decoupled Model – Phase 2

- Work spans multiple ONAP releases:
  - Honolulu:
    - Early implementation of GRA (GENERIC-RESOURCE-API) microservice
      - Primary interface used by SO

Implementing Decoupled Model – Phase 3

- Work spans multiple ONAP releases:
  - Istanbul:
    - Proof of concept version of GRA
      - Will support basic set of GENERIC-RESOURCE-API data elements:
        - Services
        - Networks
        - VNFs
        - VF-Modules

Proof of Concept – GRA microservice



- GRA microservice implements subset of 2 interfaces:
  - SLI-API: implements healthcheck
  - GENERIC-RESOURCE-API : primary interface between SO and SDNC
- Neither of these subsets currently implemented require NETCONF mounts – so OpenDaylight container is not needed for this proof of concept

Lessons Learned to Date

- Running SLI outside of karaf was fairly simple
  - SLI and most of its adaptors have no direct dependencies on OpenDaylight.
- Porting northbound interfaces is harder

Porting Northbound Interface

- 2 classes of endpoints:
  - RPCs
  - CRUD operations on config and operational trees.
- Porting RPCs is fairly straightforward:
  - Mostly same application code, except for saving state data
- Porting CRUD operations is VERY labor intensive

Porting Northbound Interface CRUD Operations

- With Yangtools/MD-SAL, code for GET/PUT/POST/DELETE to config tree is generated – BUT that code needs MD-SAL.
- We were able to generate some code using swagger-codegen for our northbound interface, but we needed to CRUD operations
- This is a major limitation : we would need to implement over 2,000 methods if we wanted to implement every possible CRUD operation for GENERIC-RESOURCE-API

Next Steps

- Istanbul:
  - GRA microservice proof of concept
    - test that GRA microservice can be used in place of current SDNC to implement at least one ONAP use case
- Beyond:
  - Improve API code generation
    - Generate implementation of CRUD operations instead of requiring manual coding


LF
NETWORKING
LFN Developer & Testing Forum