Using Bazel to improve build stability and enable codebase disaggregation

Ray Milkey
Member of Technical Staff @ ONF

Why move away from Buck?

- Builds sometimes unreproducible
- Limited extension model
- ONOS used a custom fork, high maintenance cost
- Custom java code required
- No support for remote repositories
- Limited community, not much development

Why Bazel?

- Hermetic builds
- Built in support for external repositories
- Active community, well documented
- Flexible rule model allows customization without code changes
- Multi Language support (Java, Go, C, C++ currently used in ONOS)
- Internal ONF synergy with Stratum and UPAN tool chain

Hermetic builds

- Builds should be reproducible across platforms
- Host environment should not pollute artifacts
- “Sandboxing” prevents access to files not declared as dependencies
- Hermetic artifacts can be cached and shared
- Much less churn when rebuilding

References to external repositories

- Needed to support source code disaggregation
- Each workspace carries its own configuration and build rules
- Can inject Bazel build into a non-bazel repository
- Supports GitHub repositories out of the box
- Currently used for protobufs builds, npm builds for web UI

Flexible Rules

- No need to write Bazel java code - custom rules are implemented in Starlark (formerly Skylark), a python subset
- Starlark code has access to everything in a rule, including inputs, outputs, and dependencies
- All inputs and outputs must be declared to assure hermeticity
- Examples in ONOS - OSGi jar file, Swagger, Yang model, ONOS application

Still to do

- Bazel solution for ONOS application archetypes
- Implement Sonar code coverage
- Implement a web based cache to allow sharing artifacts across builds
- Investigate remote build capability

Want to Explore or Contribute?

- ONOS Website: https://onosproject.org
- ONOS GitHub: https://github.com/opennetworkinglab/onos