operator-framework / enhancements Public

<> Code · Issues · Pull requests 15 · Actions · Projects · Wiki · Securit

a6e075119c · enhancements / enhancements / sdk-external-and-pluggable-validations.md · Go to file · ...

This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

jmrodri feedback: add a blurb to Risks & Miti… Latest commit a6e0751 7 minutes ago History

1 contributor

341 lines (259 sloc) | 13.6 KB · Raw · Blame

title	authors	reviewers		
sdk-external-and-pluggable-validations	@jmrodri	@camilamacedo86	@joelanford	@bpare

SDK External and Pluggable Validations

Release Signoff Checklist

- [x] Enhancement is `implementable`
- [x] Design details are appropriately documented from clear requirements
- [ ] Test plan is defined
- [ ] Graduation criteria for dev preview, tech preview, GA

Open Questions [optional]

1. can we use scorecard to replace all of these validations?
    - scorecard uses the cluster to run the tests, these validations typically run locally or in a pipeline. They are also done before the expensive operator tests are run.
2. what would it take to convert an existing validator to an executable format? Simply add a `main.go` to wrap existing validators.
3. do we need to add `json` tags to these structs?

Summary

Today, validations used by Container Verification Pipeline (CVP) are compiled into the `operator-sdk`. Any changes to the validation rules requires a release of operator-framework/api followed by a release of `operator-sdk`. This proposal attempts to design a way where the validations can be hosted in their own repos as well as updated without requiring new releases of the `operator-sdk`.

Motivation

Every time the business changes validation rules, it requires an update to the operator-framework/api library. A release of said library, then it needs to get included into the operator-sdk. Then a release of the SDK needs to be cut in order for the new validation rule change to be usable.

This process slows down the business by having to wait for this release process to occur for what could be a very small rule change. It also causes downstream rules to require an immediate upstream operator-sdk release, which may otherwise be as far as 3 weeks away. It is difficult to explain to the community why we need a new release for a CVP need.

Having the validators external to the SDK will allow for vendor specific validators to be created allowing for greater flexibility.

Goals

- Allow validations to be updated when the business needs them to be
- Allow validations to release when the authors need them to be
- Do not require newer builds of the operator-sdk to get updated rules
- Allow validations to be hosted in their own repos

- Allow validations to be external to operator-sdk

Non-Goals

- Migrate existing validators to the new format

Proposal

User Stories [optional]

Story 1

- as a CVP admin I would like to change the validation for XXX and have it useable as soon as I can do a validator release.

Story 2

- as a validation author, I can write a validation for a bundle and use it without having to get a new operator-sdk release

Story 3

- Users want to edit/add/remove a set of validation rules for Operator bundle validation.

Story 4

- Users can host a set of validation rules either locally or in remote.

Story 5

- Users can define the target set of validation rules for SDK's bundle validation command (either from local or remote source).

Implementation Details/Notes/Constraints [optional]

There are a few alternatives, but I think wrapping the validations in their own executables makes sense. It aligns with the work we've done with Phase 2 plugins. It allows the most flexibility in terms of implementation. It also has a simple API - input: bundle dir; output: ManifestResult JSON.

Wrapping validations in their own executable simply means that there needs to be something that the operator-sdk can run like a shell script or a binary.

The validation executable will need to accept a single input value which is the bundle directory. The output will need to be a ManifestResult JSON. This JSON will be parsed by the operator-sdk converted and output as a Result.

For example, the existing validations could easily be wrapped with a `main.go` file and compiled into a binary. They could be copied to their own repos for easier release. Now you might be thinking that this means we'll have many releases still, we actually would only have one (1) release which is the validator itself. For go based validators this might mean a new binary, but you could also release your validator in python and make it a single file or even a shell script.

From the operator-sdk's point of view, we don't care what you use to create your validator only that we can run it and pass it a bundle directory.

Validators

Running validators

The existing validators operate on a bundle. Each validator should be some executable that accepts a single argument, the bundle root directory. So `operator-sdk` will pass them the bundle directory. It will be the validator's responsibility to parse the bundle to get what it needs.

For example, the validator executable will be invoked from the `operator-sdk` in the following manner:

```
/path/to/validator/executable /path/to/bundle
```

A concrete example might look like this:

```
./validator-poc /home/user/dev/gatekeeper-operator/bundle
```

The actual implementation is up to the author. Here we have an example of the `OperatorHubValidator` as a Go binary.

The validators can be written in any language but there must be an executable entry point that accepts a single bundle path as a CLI argument. For example, you can write your validator in python but you would want to make the main python file executable or supply a shell script to be invoked.

Validator results

As stated earlier, each validator should be some executable that accepts a single argument, the bundle root directory. Th validators should also output ManifestResult JSON to stdout.

Because the existing validators currently return a ManifestResult, it seems logical that we use the same object as JSON for the output.

The validator executable should exit non-zero ONLY if the entrypoint failed to run NOT if the bundle validation fails.

For example, let's say the validator is given a path to the gatekeeper bundle. The validator should validate the given bundle and output the ManifestResult JSON. Here is an example of a run:

```
{
    "Name": "gatekeeper-operator.v0.2.0-rc.3",
    "Errors": null,
    "Warnings": [
        {
            "Type": "CSVFileNotValid",
            "Level": "Warning",
            "Field": "",
            "BadValue": "",
            "Detail": "(gatekeeper-operator.v0.2.0-rc.3) csv.Spec.m
        }
    ]
}
```

The above JSON will be read by the `operator-sdk` during `bundle validate` command and output the results as it does today. The example below shows what `operator-sdk bundle validate` would printout if given the ManifestResult from above.

```
{
    "passed": true,
    "outputs": [
        {
            "type": "warning",
            "message": "Warning: Value : (gatekeeper-operator.v0.2.
        }
    ]
}
```

Allowing the validators to output ManifestResult should make it easier to transition existing validators to the external format with minimal code.

Migrating existing validator to executable

In the short to near term, you create a new `main.go` for each of the validators. Then import the validator code from operator-framework/api. The `main.go` would take in one (1) argument, the bundle directory.

Since the existing validators already output `ManifestResult`, it's easiest if we simply print that out as JSON to stdout.

An example POC that takes an existing validator and outputs `ManifestResult` can be found at validator-poc

Another example of a migration can be find at ocp-olm-catalog-validator. This particular example does NOT yet output `ManifestResult` format.

CLI

The big question at hand is how do we indicate to the `operator-sdk` CLI that we want to run an external validator? Today, the `bundle validate` command takes in a few flags, here we will discuss how each might need to be changed to work with external validators.

```
Usage:
  operator-sdk bundle validate [flags]

...

Flags:
      --alpha-select-external string
Selector to select external validators to run. It should be set
to a Unix path list ("/path/to/e1.sh:/path/to/e2")
  -h, --help                                                help
for validate
  -b, --image-builder string                                Tool
to pull and unpack bundle images. Only used when validating a
bundle image. One of: [docker, podman, none] (default "docker")
      --list-optional                                       List
all optional validators available. When set, no validators will
be run
      --optional-values --optional-values=k8s-version=1.22
Inform a []string map of key=values which can be used by the
validator. e.g. to check the operator bundle against an
Kubernetes version that it is intended to be distributed use
--optional-values=k8s-version=1.22 (default [])
  -o, --output string
Result format for results. One of: [text, json-alpha1]. Note:
output format types containing "alphaX" are subject to change and
```

```
not covered by guarantees of stable APIs. (default "text")
      --select-optional string
Label selector to select optional validators to run. Run this
command with '--list-optional' to list available optional
validators

Global Flags:
      --plugins strings   plugin keys to be used for this
subcommand execution
      --verbose           Enable verbose logging
```

- *--help* works as is
- *--image-builder* works as is
- *--list-optional* would need to be updated to locate external validators.
- *--optional-values* would continue to be passed to the validators
- *--output* indicates how we want to output the results.
- *--select-optional* works as is
- -- *alpha-select-external* added; takes in the location of the executable to run as the validator, i.e. `/path/to/validator/the-executable:/path/to/another`

Risks and Mitigations

There is little risk, if this does not pan out we keep going on the current path of compiling them in.

Another possible risk is users of an airgapped environment will have to install these external validator executables. We could supply a mechanism for distributing these external validators as an image. For this first release, we will ignore the distribution solutions leaving it to the user to copy the validators onto the system running `operator-sdk`.

Design Details

Test Plan

- unit tests for the feature will be added in the implementation
- QE would need to create custom validations to be called by the new `--alpha-select-external` flag.

Graduation Criteria

This feature will come out as alpha accessible via the `--alpha-select-external` flag to the `bundle validate` command. We will accept feedback on possible changes to the feature.

After a period of time, we can graduate to a GA with a more permanent flag, i.e. `--select-external`.

Version Skew Strategy

- The version of the validators can change however the validator author sees fit.
- The API or contract between `operator-sdk` and validators will be
  - input to validator: bundle directory
  - output from validator: `ManifestResult` JSON

Implementation History

N/A

Drawbacks

Validators would have to be their own executables which could result in a compilation step being needed depending on the language used to implement them.

Alternatives

- put validations in their own images
  - need to define "API" contract what is the entrypoint and what parameters do we give it
  - pro:
    - already have a precendence for running things from images
    - familiar tech
  - con:
    - would need a cluster to run these validations
    - authors would have to create binaries of their validations anyway
- use a language like JavaScript or CUE to define all validations

  - validations could be run from a git repo, i.e. operator-sdk could pull it and then evaluate it
  - pro:
    - simpler delivery, expose via a gitrepo and done
  - con:
    - all existing validations would have to be re-written in a new language structure which could introduce new bugs
    - unproven technology
    - would have to write the engine to know how to execute these

- use scorecard to do the validations
  - create validations written in scorecard as custom tests
  - pro:
    - infrastructure required to run is already built within scorecard
  - con:
    - would need a cluster to run these validations

Infrastructure Needed [optional]

N/A