Crossplane

Testing and Release Patterns for Crossplane

Stefano Borrelli and Yury Tsarev, upbound

August 22th 2024

KubeCon | CloudNativeCon

THE LINUX FOUNDATION OPEN SOURCE SUMMIT

AI_dev Open Source GenAI & ML Summit

China 2024

Stefano Borrelli

Principal Solutions Architect

Stefano has worked as a Systems Engineer for the finance and technology companies, as well as starting a cloud infrastructure startup. He joined Upbound to drive adoption of Control Planes in the enterprise and is a contributor to the Crossplane project.

Crossplane

upbound

Yury Tsarev

Principal Solutions Architect

An active contributor to the Kubernetes and Crossplane ecosystems, Yury has worked with leading firms in Linux distribution, big data, video streaming, consulting, and finance. Yury joined Upbound to spearhead the control plane revolution in cloud-native infrastructure.

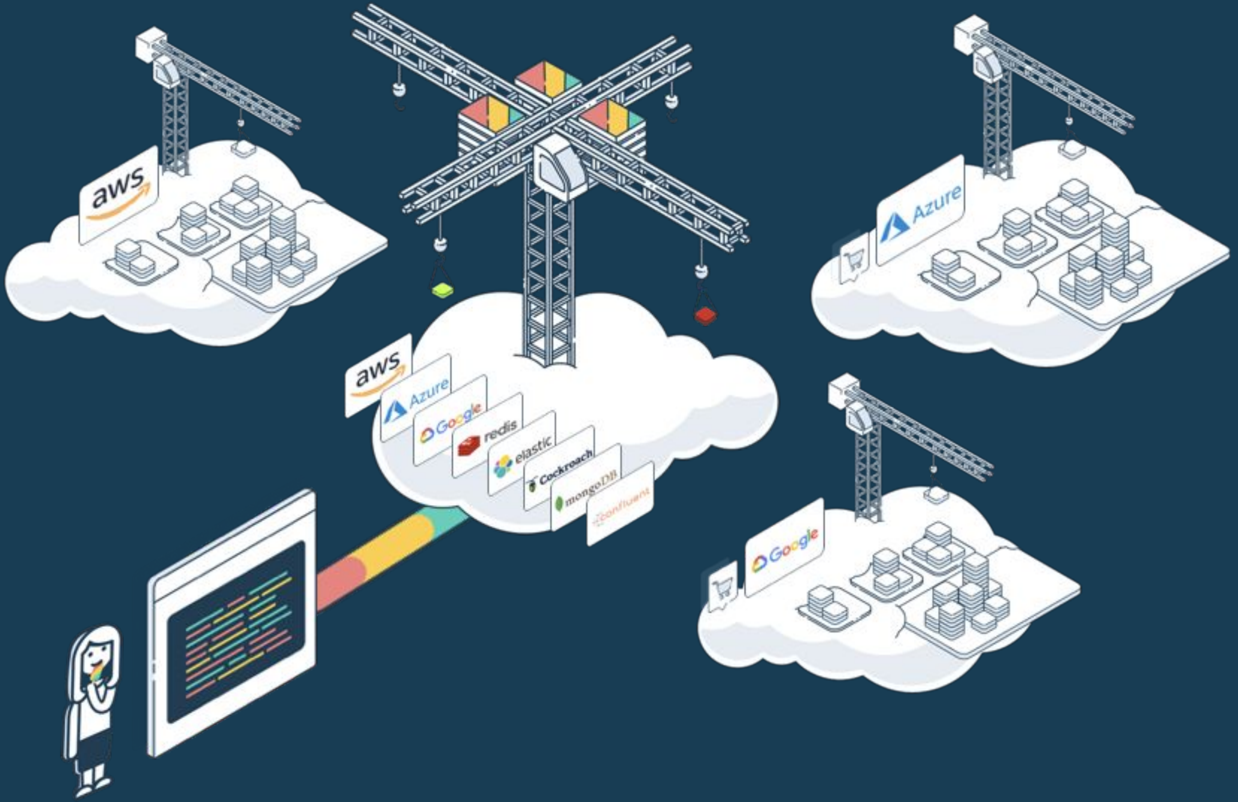
Crossplane

upbound

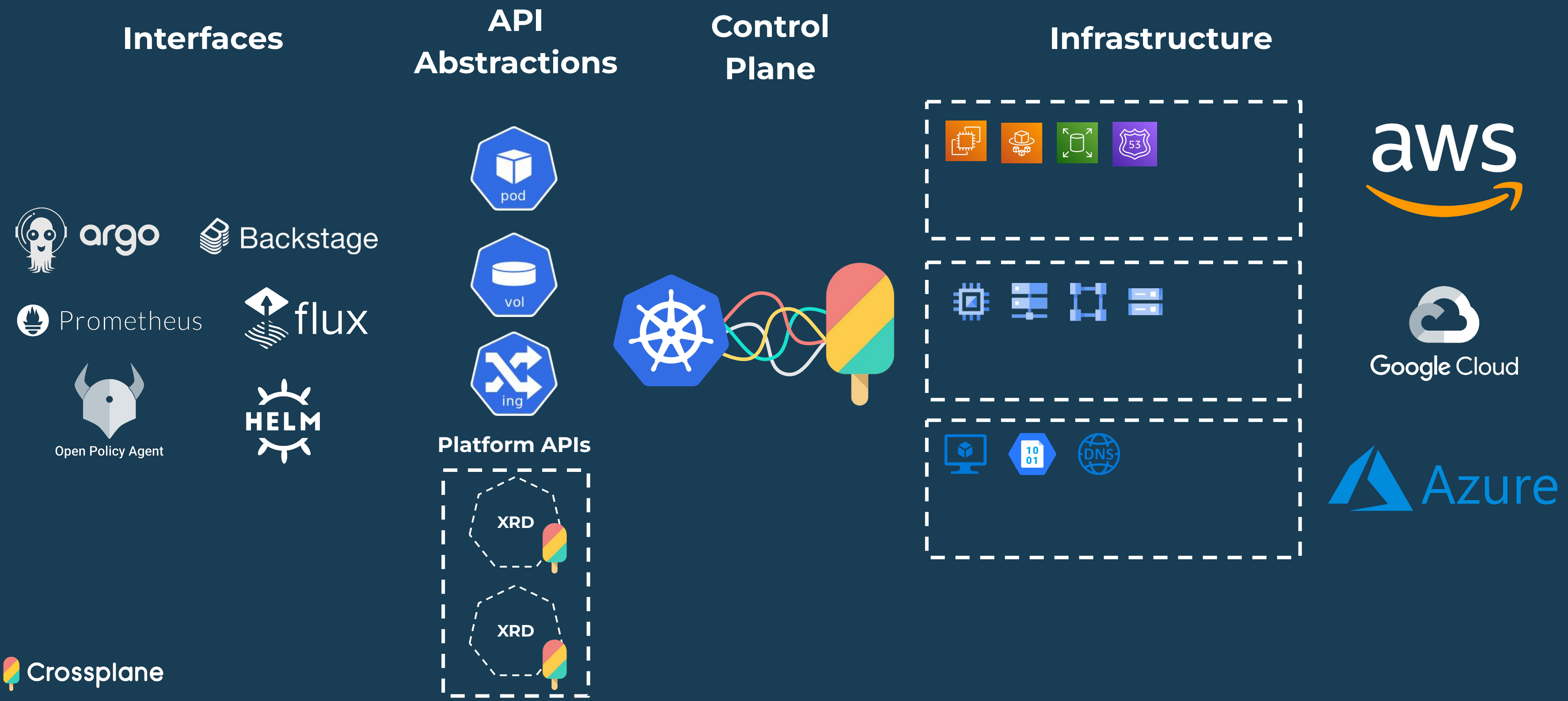
Introduction
To
Crossplane

Crossplane is a Kubernetes-based Universal Control Plane that lets you manage **anything** that has an API.

With a strong API and Extension system it is ideal for building **internal cloud platforms.**



The Universal Control Plane



Crossplane

Core Concepts

Providers

Controllers + CRDs to Support any API

Composition

Define a custom API and combine Resources.

Functions

Write desired state in any programming language.

Core Concepts: Testing Goals

Providers

Controllers + CRDs to Support any API

Composition

Define a custom API and combine Resources.

Functions

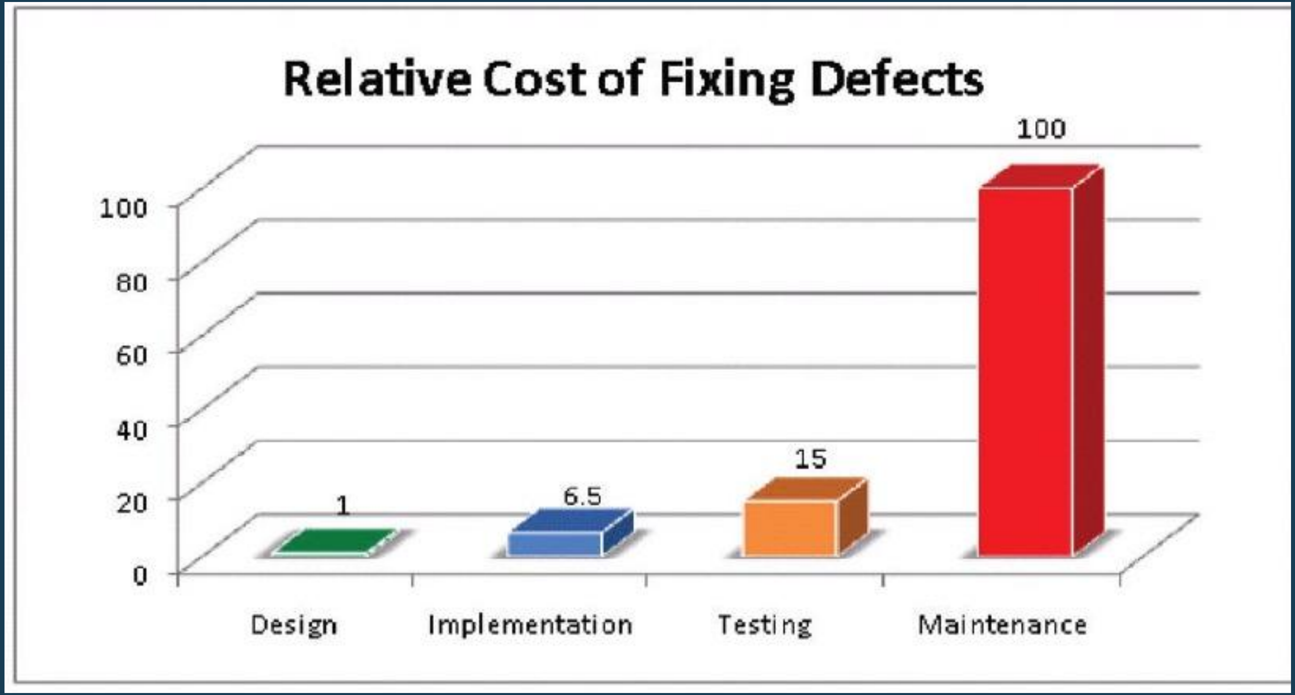
Write desired state in any programming language or text template.

Validate Desired Resources against Schema

Validate Desired State of Resources

Function or Template logic is correct

The Importance of Catching Errors Early



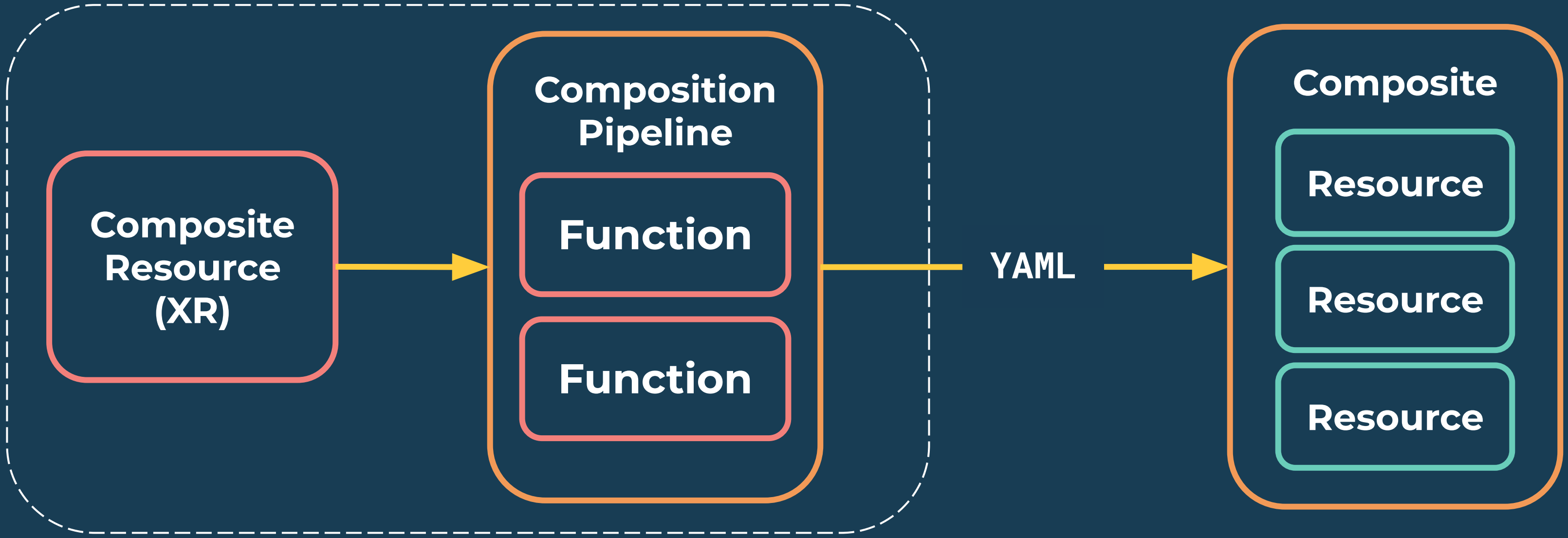
Crossplane

[Dawson, Maurice & Burrell, Darrell & Rahim, Emad & Brewster, Stephen. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC).]

Rendering Compositions Locally

Rendering Functions

We can use Functions and the Crossplane CLI using **render** to simulate how Compositions would be rendered on a Crossplane Cluster.



```
crossplane render xr.yaml composition.yaml functions.yaml
```

Crossplane

crossplane render

crossplane render requires a few arguments. In Crossplane 1.17, the **`beta`** argument will be removed.

```
1.16  crossplane beta render xr.yaml composition.yaml functions.yaml <args>
1.17+ crossplane render xr.yaml composition.yaml functions.yaml <args>
```

Composite Resource

Composition to render

Function packages in the pipeline

Crossplane

upbound

Mocking Crossplane Clusters

Composition Data Types

Environment Configs

A ConfigMap-like CRD that can be used to store structured data. See function-environment-configs.

Observed Data

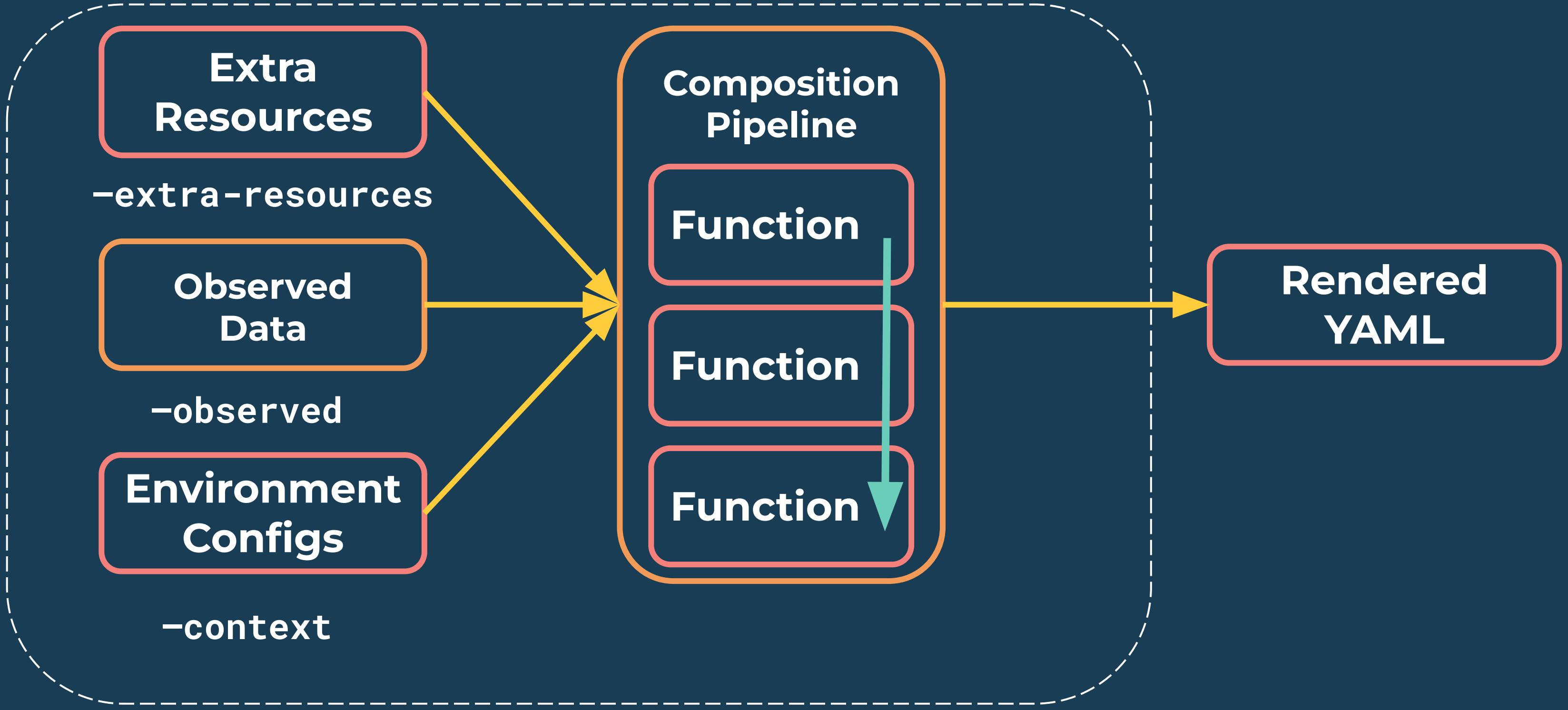
Existing Composition Resources observed by Crossplane. `kubectl get -o yaml` to generate these files for testing.

Extra Resources

Allows Functions to pull in any information Crossplane has access to using a Selector. See function-extra-resources. Can include Environment Configs, Managed Resources, or any other Crossplane object.

Simulating External Resources

```
crossplane render <args>
```



Crossplane

Simulating Observed Data

Using the CLI options, we can add Observed and Extra data to our run. The `--include-full-xr` option includes the **`spec`** from the Composite xr.yaml.

```
crossplane beta render \
  --observed-resources observed/ \
  --extra-resources environment/dev.yaml \
  --include-full-xr \
  xr.yaml composition.yaml functions.yaml
```

Validating Render Outputs And Other Manifests

Validating Function Output

The validate option in the crossplane CLI allows us to validate generated manifests against CRDs. Schemas can be downloaded from Crossplane packages.

```
crossplane beta validate <args>
```

Validate can be used in conjunction with **`render`**:

```
crossplane beta render \
  --extra-resources environment/dev.yaml \
  --include-full-xr \
  xr.yaml composition.yaml functions.yaml | crossplane beta
validate schemaDir -
```

Demo: Render and Validate

https://github.com/upbound/composition-testing/tree/main

Unit Testing with KCL

KCL and function-kcl

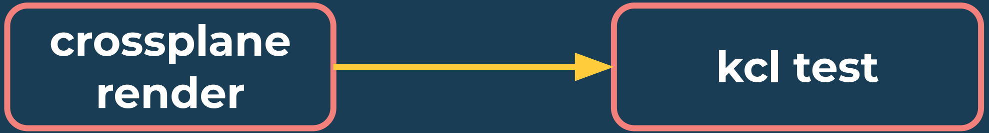
- https://www.kcl-lang.io/
- https://github.com/crossplane-contrib/function-kcl amazing contribution to Crossplane ecosystem coming from China
- KCL/function-kcl intros
  - https://blog.crossplane.io/function-kcl/ by maintainer
  - https://blog.upbound.io/kcl-benefits-crossplane-composition-building by Upbound showcasing KCL in reference configuration-aws-eks

KCL Playground

- Available both online and locally as `kcl play` cli
- Ability to quickly prototype logic even before the code commit
- Ability to share the prototypes and fix issues quickly

Unit testing with KCL

- What is unit in our case?
  - We could test unit test `*.k` files but on this level we have limited crossplane environment mocking capabilities
  - We will test crossplane render state instead
  - This way we can test any function and whole function pipeline
- Assertions against `crossplane beta render` yaml stream
- Fast feedback loop
- Possibility to create TDD-like development flow for Composition crafting
- Example: https://github.com/upbound/configuration-azure-network/pull/55
- DEMO



End-to-End Testing

The Role of End-to-End (E2E) Testing

- Local rendering and validation is great but we always need a real acceptance test
- We have to perform manual E2E test before creating a Pull Request
    - Apply/Package XRD and Composition
    - Apply Claim/XR
    - Evaluate expected resource behavior
        - is it Synced?/Ready?/Deleted?
- We want to automate all the manual steps
- We want organically growing regression test suite
- High confidence before Configuration release

Introducing Uptest

- Low code E2E test automation framework
- Can be used for testing of
  - Crossplane Providers
  - Crossplane Configurations
- Originally created by Upbound to support development velocity and reliability of official providers
- Eventually used for testing of all reference Configurations
- Recently contributed to CNCF https://github.com/crossplane/uptest

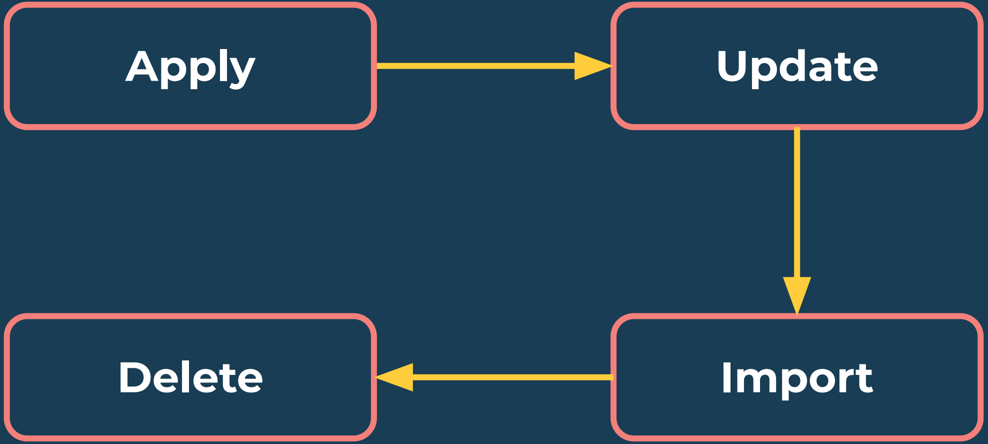
Uptest Capabilities

- Uptest will automatically generate the KUTTL test suite for you in the background on the fly
  - Just example manifest as input
  - No KUTTL test code assertions to maintain
- Standard location for Examples is `examples` directory for the Crossplanes repository containing Provider or Configuration, e.g. https://github.com/upbound/configuration-azure-network/tree/main/examples

Uptest Testing Steps

Performed Test Steps can be reviewed by associated KUTTL suite templates at https://github.com/crossplane/uptest/tree/main/internal/templates
It is full CRUD-like flow with Crossplane-specific Update and Import steps

1. Apply
2. Assert
3. Update
4. Post Update Assert
5. Import
6. Post Import Assert
7. Delete



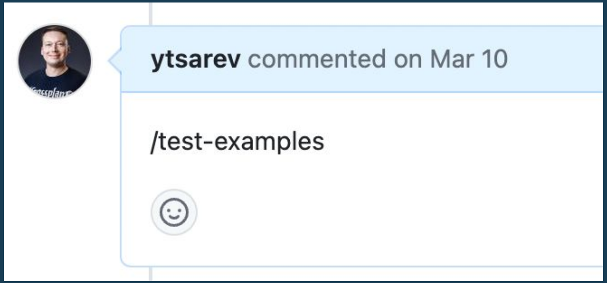
Uptest Hooks and Helper Functions

- Hooks
    - Setup
    - Teardown
    - PreAssert
    - PostAssert
    - PreDelete
    - PostDelete
    - Controlled by annotation in example manifest, e.g.
      `uptest.upbound.io/pre-delete-hook: testhooks/delete-apikey.sh`

- Helper Functions
    - Injecting arbitrary values to the manifests with `${data.key}`
    - Randomize `metadata.name` with `${Rand.RFC1123Subdomain}`

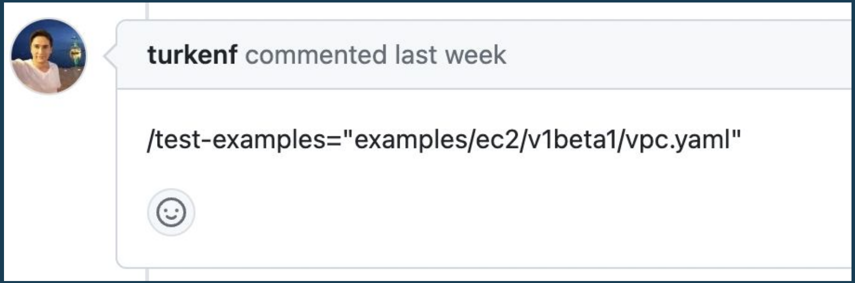
Uptest CI Pipeline Integration

- https://github.com/upbound/official-providers-ci the place for different CI tooling and integration Upbound uses to support official providers pipelines
- Reusable Github Actions workflow for running Uptest in a centralized manner https://github.com/upbound/official-providers-ci/blob/main/.github/workflows/pr-comment-trigger.yml
- Easy to integrate with any GitHub repository
- On-demand centralized uptest run with PullRequest comment



ytsarev commented on Mar 10

/test-examples



turkenf commented last week

/test-examples="examples/ec2/v1beta1/vpc.yaml"

Uptest Demo

- Sample Configuration repo https://github.com/upbound/configuration-azure-network
- Look at examples directory
- Investigate uptest target
- Configure credentials
- Run uptest locally
- Run uptest in CI with PR comment

Uptest Upcoming Enhancements

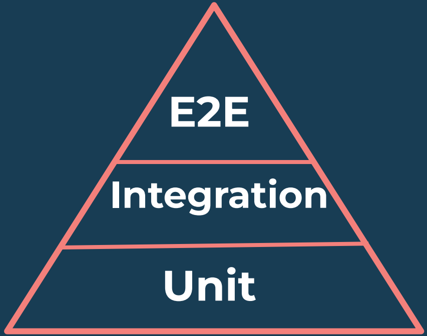
- Switch from KUTTL to Chainsaw https://github.com/kyverno/chainsaw
    - Proposal https://github.com/upbound/official-providers-ci/issues/179
    - Maintainer One-Pager https://github.com/crossplane/uptest/pull/11
    - PR: https://github.com/crossplane/uptest/pull/15
- Chainsaw is a successor to KUTTL
    - Much more powerful assertions
    - Better logs
    - Full backward compatibility
- Check out Viktor's Video on Chainsaw!

Try Uptest!

- Clone sample Configuration repository
  - https://github.com/upbound/configuration-azure-network
- Investigate relevant Make targets of uptest and e2e
  - https://github.com/upbound/configuration-azure-network/blob/main/Makefile#L70-L80
- Configure cloud credentials
- Run make submodules to pull Crossplane build module
- Run make e2e
- Transfer this setup to your Crossplane Configuration repository
- You are ready to go!

Leave us feedback at https://github.com/crossplane/uptest/issues

Recap



Gradually proceed from local testing to e2e according to test pyramid

- Use local fast-feedback techniques like `render` and `validate`
- Apply TDD-flow to develop your infrastructure
- When you have enough signals from local tools, proceed with expensive e2e testing in automated way using `uptest`

Learn more!

- Reuse testing and other best practices we gathered in a set of reference platforms
  - https://github.com/upbound/platform-ref-aws/
  - https://github.com/upbound/platform-ref-azure/
  - https://github.com/upbound/platform-ref-gcp/
- Check recent blogposts
  https://blog.upbound.io/crossplane-aws-platform-configuration
- https://blog.upbound.io/composition-testing-patterns-rendering

Crossplane

Thank you!

Slides available at sched link below!
Please also submit feedback there!

https://sched.co/1eYZ7

upbound