

Crossplane

Contribfest

<https://crossplane.io>

Jared Watts, Muvaffak Onuş - Speakers
Yury Tsarev, Christopher Haar - Moderators





Agenda, Goals, & Tracks

- Grow and empower a high quality contributor base
 - All projects can benefit from more great contributors
- How to Write & Test code in Crossplane
 - Walkthrough the development workflow
 - Hands-on lab to make a code change and test it
- How to Write a Composition Function
 - Accelerate your understanding and adoption of this powerful new alpha feature in v1.11

Contributing to Crossplane



Let's kick off a build!

- This may take a bit... 🕒
- Pre-Requisites
 - docker client is running
 - go is installed
- Run these commands
 - `git clone https://github.com/crossplane/crossplane.git`
 - `cd crossplane`
 - `make`
- Long poles
 - go module dependencies
 - OCI image layers

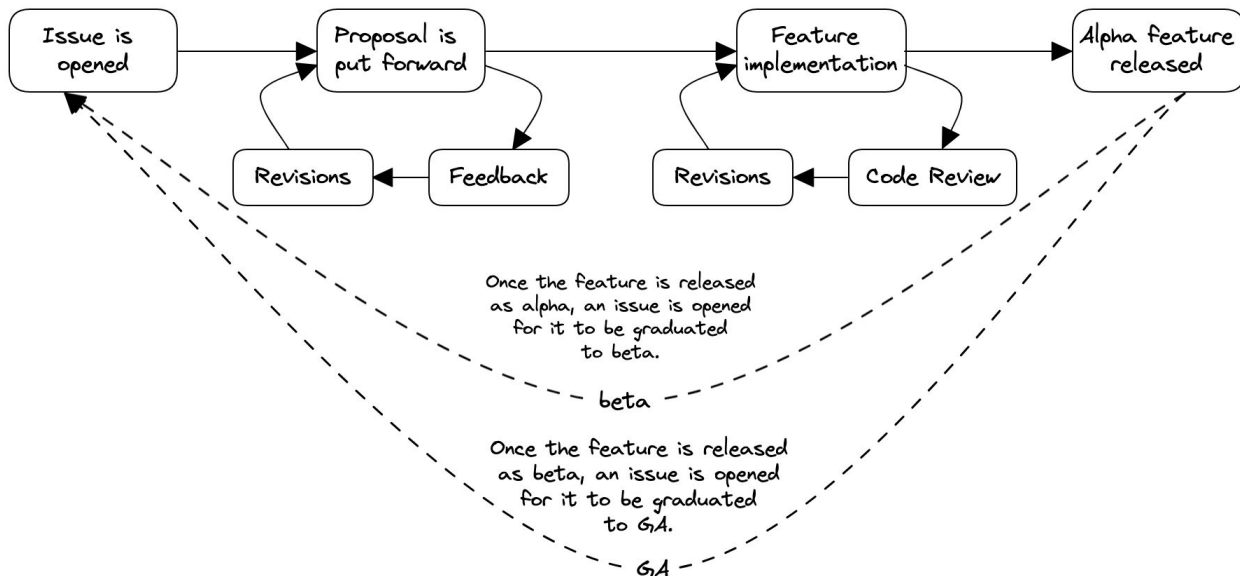


Contributor Resources

- [Contributing folder](#) in crossplane/crossplane repo
- Main [README.md](#) has a full contributor guide
 - contributions always accepted to clarify/improve 😊
 - contribute to the contributing guide - contribution inception!
- [Release process](#) - how to release new versions of Crossplane
- [Provider development guide](#) - how to design and build your own custom providers
- [Observability guide](#) - how to add events, logging, metrics to your Crossplane core controllers and providers
- And more!

The Life of a Feature

Development cycle of a new feature



- [Feature life cycle docs](#)



Release Cadence and Cycles

- Quarterly releases - 3 month cycle, 4 releases per year
- Development → Feature Freeze → Code Freeze → Release
- Last 3 versions are maintained with patch releases
 - [Current releases table](#)
 - e.g. v1.12, v1.11, v1.10
 - Backport PR automation with `backport release-x.ylabel`
- [Release cadence and cycles docs](#)



Contributor Workflow

- [Contributing code workflow docs](#)
- Communicate your intent
 - Discuss your change in a GitHub issue **before** you start
- Make your changes
 - Tell a story with your git commits and always rebase off of upstream
 - [Sign-off](#) on all Git commits by running `git commit -s`
 - Preempt [coding style](#) review comments
- Test your changes
 - Add or update test cases for all changes
- Update documentation and examples where appropriate
- Open a Pull Request (PR)
 - Fill out entire [PR template](#)
 - Don't force push while addressing review feedback, but do tidy up your commit history before merging



Components and Codebase

- [crossplane-runtime](#) - reusable logic across core and providers
 - [Managed Reconciler](#) - core reconciliation logic for managed resources
 - [Errors, Events, Logging](#)
 - [Metadata operations](#)
 - [Resource model](#)
- [crossplane](#) - core crossplane logic
 - [main entry points](#)
 - [API definitions](#)
 - Controllers - [Compositions](#), [Packages](#), [RBAC](#)



Make builds Crossplane

- All of Crossplane's build logic is implemented in make
- Crossplane specific [Makefile](#) and reusable [build submodule](#)
- `make build` - build all artifacts
- `make reviewable` - (generate, lint, test)
- `make run` - run Crossplane directly in proc
- `make` is called from Github Actions workflows for official CI and releases, e.g. [build](#), [test](#), and [promote](#)



Build Artifacts

- OCI images, e.g.:
 - [crossplane/crossplane:v1.11.3](#)
 - [crossplane/xfn:v1.11.3](#)
- Releases bucket (binaries) - [https://releases.crossplane.io/stable/v1.11.3/bin/](#)
- Helm charts bucket - [https://charts.crossplane.io/stable/](#)

Hands-on Lab 1

Writing & Testing Code in Crossplane



Lab 1 Content

- Content can be found in [crossplane-contrib/contribfest](https://github.com/crossplane-contrib/contribfest) repo
- Overview
 - Set up the dev environment
 - Build Crossplane artifacts
 - Set up a local cluster and deploy your changes
 - Make a code change, rebuild artifacts and deploy your changes
 - Running unit tests and integration tests
 - Getting changes ready for review with `make reviewable`
 - Opening a PR with your changes



Lab 1 - Let's go!

Hands on the keyboard!  

Follow along with the Lab 1 content in the contribfest repo:
[crossplane-contrib/contribfest](https://github.com/crossplane-contrib/contribfest)



Coding Standards

- Contributing guide's [coding style](#) section
- We're mostly following common standards in Go
 - Follow the guidelines set out by the [Effective Go](#) document
 - Preempt common Go [code review comments](#) and [test review comments](#)
- Follow Crossplane's [Observability Developer Guide](#)
 - How to use error reporting, logging, events, metrics, etc. within your code
- [Explain `noLint` directives](#)
- [Use descriptive variable names sparingly](#)
- [Don't wrap function signatures](#)
- [Return early](#)
- [Wrap errors](#), [Scope errors](#)
- [Prefer table driven tests](#)



PRs done well

- PRs are a sensitive two sided engagement for authors and reviewers
 - Approach with kind collaboration
 - Both sides need to understand intent and context
 - Conversational style feedback, not terse instructions
 - Asking questions and suggesting changes, rather than demanding
 - Be clear about blockers and nice to haves
 - We're all here to make the project better 🤝
- Clean commit history with a [rebase workflow](#)
 - Create your own fork and branch for your work
 - When updating with new changes, always rebase your fork/branch off upstream
- Full [code review process](#) explained in contributing guide

Building a Composition Function

Why?

```
apiVersion: database.starter.org/v1alpha1
kind: XPostgreSQLInstance
metadata:
  name: my-db-hs7dy
spec:
  parameters:
    region: east
    size: small
    storage: 20
  claimRef:
    name: my-db
    namespace: default
```

Composition
Resource

```
apiVersion: rds.aws.upbound.io/v1beta1
kind: Instance
metadata:
  name: my-db-hs7dy-osu78
spec:
  forProvider:
    allocatedStorage: 20
    autoGeneratePassword: true
    autoMinorVersionUpgrade: true
    backupRetentionPeriod: 14
    backupWindow: 09:46-10:16
    engine: postgres
    engineVersion: "13.7"
    instanceClass: db.t3.micro
    maintenanceWindow: Mon:00:00-Mon:03:00
    name: example
    passwordSecretRef:
      key: password
      name: example-dbinstance
      namespace: upbound-system
    publiclyAccessible: false
    region: us-east-1
    skipFinalSnapshot: true
    storageEncrypted: true
    storageType: gp2
    username: adminuser
  writeConnectionSecretToRef:
    name: example-dbinstance-out
    namespace: default
```

Why?

```
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: xpostgresinstances.aws.database.starter.org
spec:
  writeConnectionSecretsToNamespace: upbound-system
  compositeTypeRef:
    apiVersion: database.starter.org/v1alpha1
    kind: XPostgreSQLInstance
  resources:
    - name: rdsinstance
      base:
        apiVersion: rds.aws.upbound.io/v1beta1
        kind: Instance
        spec:
          forProvider:
            region: us-east-1
            instanceClass: db.t2.small
            username: adminuser
            engine: postgres
            engineVersion: "12"
            skipFinalSnapshot: true
            publiclyAccessible: false
            autoGeneratePassword: true
            passwordSecretRef:
              namespace: upbound-system
              key: password
          writeConnectionSecretToRef:
            namespace: upbound-system
```

```
patches:
  - fromFieldPath: "spec.parameters.region"
    toFieldPath: "spec.forProvider.region"
    transforms:
      - type: map
        map:
          east: us-east-1
          west: us-west-1
  - fromFieldPath: "spec.parameters.size"
    toFieldPath: "spec.forProvider.instanceClass"
    transforms:
      - type: map
        map:
          small: db.t2.small
          medium: db.t2.medium
          large: db.t2.large
  - fromFieldPath: "spec.parameters.storage"
    toFieldPath: "spec.forProvider.allocatedStorage"
```



Limitations of Composition

- No conditionals or for loops.
- List of resources is always the same.
- No advanced logic other than simple transforms.
- Random values to be generated only once.
- No ability to call external APIs to get values.
- ...
- It is not a programming language.



Composition Functions

- Released as alpha in [v1.11.0](#)
- Any container that can receive and return a FunctionIO object would work - no language limitation.
- Additive to the existing composition mechanisms.
- Pipeline of functions allow modularity.
- Configuration can be supplied through ``Composition``.
- ...
- All the things you couldn't do with Composition!



Examples

- Render with Go templates (written in Go)
 - <https://github.com/negz/xfns/pull/1>
- Add a quote to every managed resource (written in Python)
 - <https://github.com/negz/xfns/pull/2>
- No-op function (written in Bash)
 - <https://github.com/negz/xfns/tree/main/xfn-nop>
- ...
- The ones we will build!
 - Make sure to add them to the list in [contribfest](#) repo readme!

Back to keyboard!

Open <https://github.com/crossplane-contrib/contribfest>

Community is everything



Get Involved

- Website: <https://crossplane.io/>
- Docs: <https://docs.crossplane.io/>
- GitHub: <https://github.com/crossplane/crossplane>
- Slack: <https://slack.crossplane.io/>
- Blog: <https://blog.crossplane.io/>
- Twitter: https://twitter.com/crossplane_io
- Youtube: [Crossplane Youtube](#)