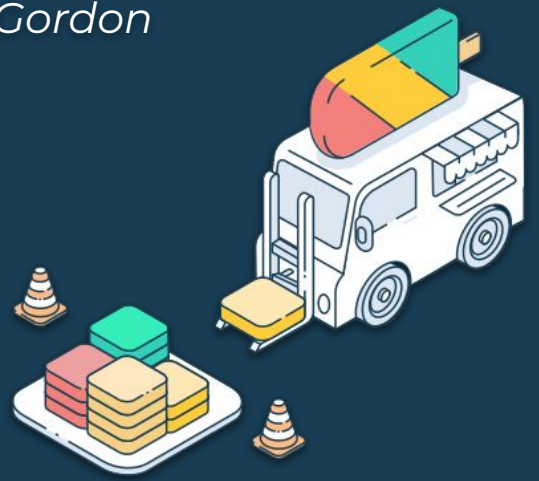
Crossplane

The Cloud-Native Framework for Platform Engineering

Jared Watts, Adam Wolfe Gordon
Maintainers



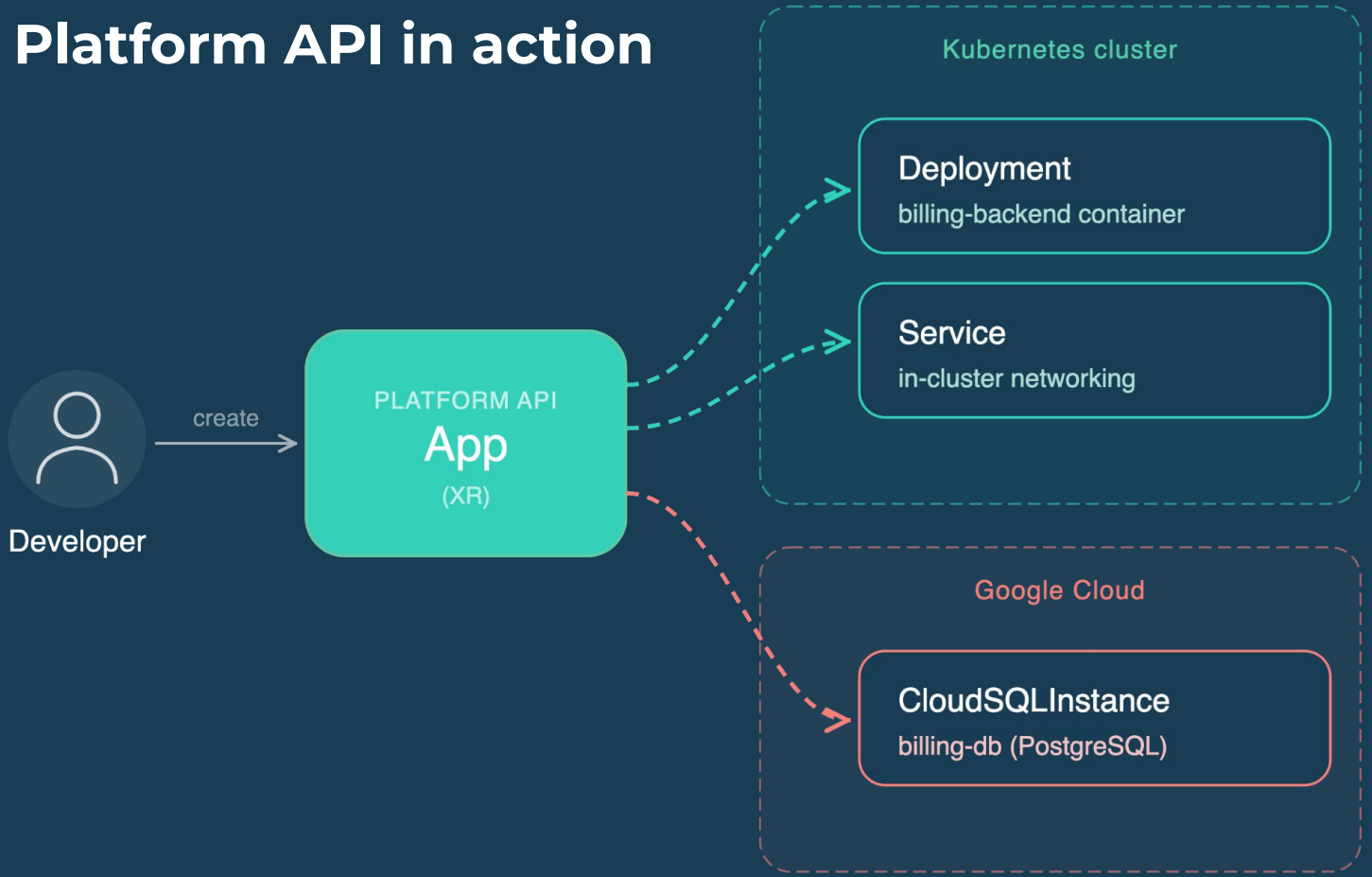
A Platform Story...

- A developer builds a new service, time to deploy!
- They need infrastructure — database, cache, message queue, DNS, certificates...
- Compliance, security, policy, encryption, replication, high availability, best practices, cost optimization, tagging...
- They file a ticket and wait.
- Days. Maybe weeks.
- Need to ship, but they’re blocked by process and complexity

Platform APIs - a better way

```yaml
apiVersion: platform.example.com/v1
kind: App
metadata:
  namespace: team-billing
  name: billing-backend
spec:
  image: registry.example.com/team-billing/backend:v1.21.2
  requirements:
  - Database
  database:
    engine: Postgres
    storageGB: 100
 status:
   endpoint: https://backend.billing.platform.example.com/api/v1
```

Platform API in action



Crossplane is a control plane framework

- **Self-service, with guardrails**
  - Your API frames the platform - expose only what users need
- **Declarative APIs**
  - Describe what you want, not how
- **Continuous reconciliation**
  - Always driving observed -> desired state, correcting drift
- **Kubernetes is a control plane**
  - Great at orchestrating containers, what about managing **all** your resources?

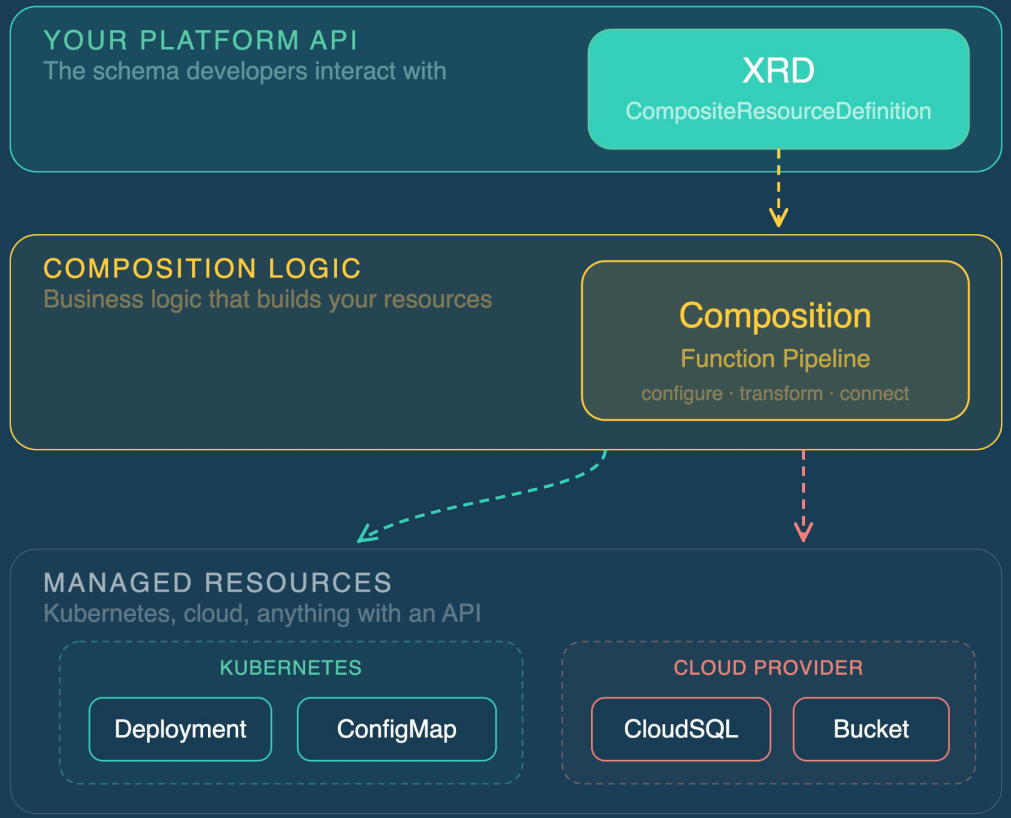
Crossplane is now a CNCF Graduated project! 🎓

- 3,000+ contributors - top 10% of all CNCF projects
- 500+ companies contributing
- 140+ releases
- Lots of adopters in **production** and at **scale**
  - Nike, Autodesk, Grafana, Elastic, Akamai, SAP, IBM, Nokia, etc.

What's new in Crossplane v2?

- Crossplane v2 is **backward compatible with v1.x**
- Big changes
  - **Namespaced resources** - XRs and MRs live in namespaces
  - **Compose anything** - Not just MRs, any Kubernetes resource
  - **Less opinionated** - Use just MRs, just XRs, or both
  - **No more claims** - Simpler, less obfuscation
  - **Activation Policies** - Install only the CRDs you need
- Crossplane v1.20 still supported - critical fixes only

Anatomy of a Platform



Managed Resources

Building Blocks

Crossplane

Managed Resources Example: AWS

Networking

Databases

Kubernetes Clusters

IAM

VMs

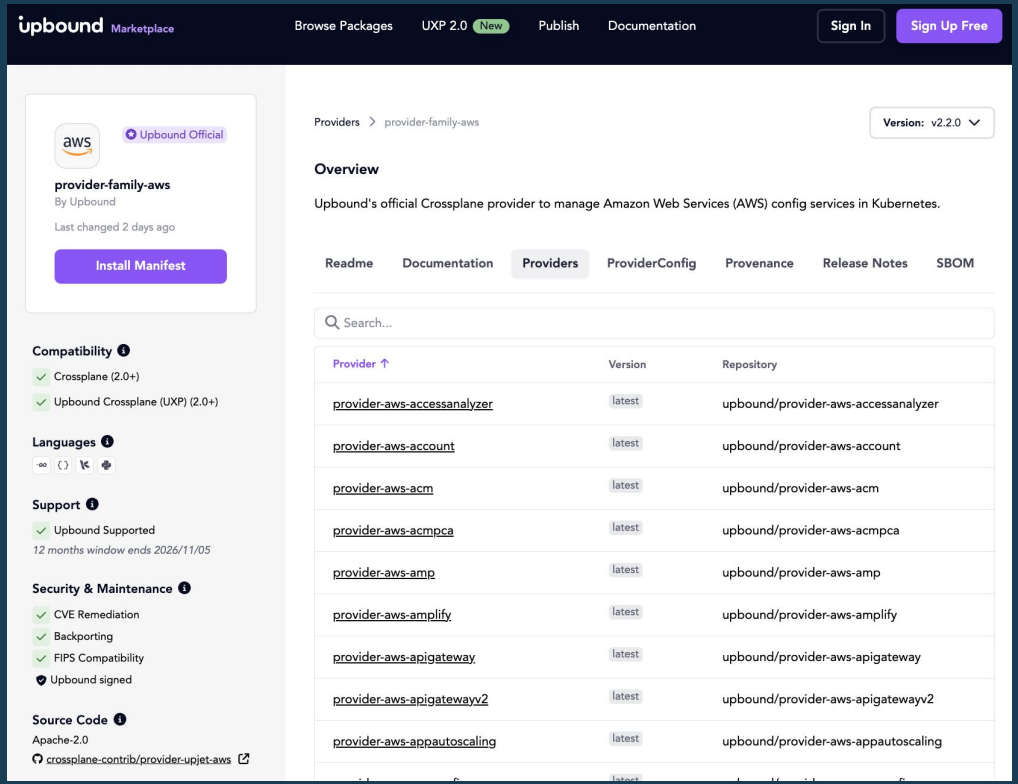
AI/ML

Message Queues

Caches

Certificates

...and much more...

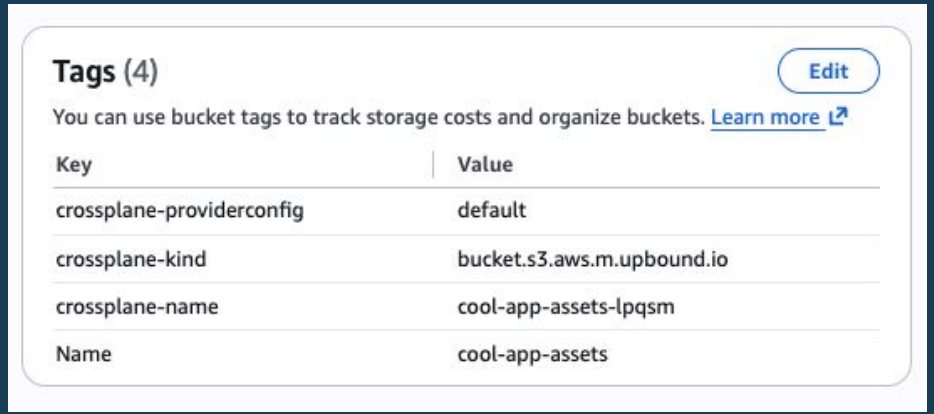


https://github.com/crossplane-contrib/provider-upjet-aws

Crossplane

Managed Resources

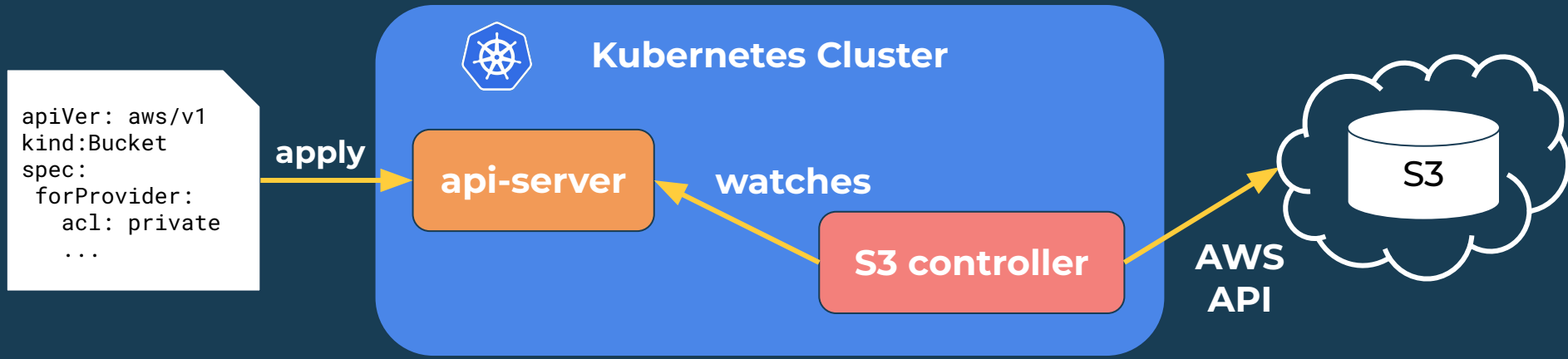
```yaml
apiVersion: s3.aws.m.crossplane.io/v1beta1
kind: Bucket
metadata:
  generateName: cool-app-assets-
  namespace: cool-app
spec:
  forProvider:
    region: us-west-1
    objectLockEnabled: true
    tags:
      Name: cool-app-assets
```



Crossplane

Managed Resource Reconciliation

- Controllers reconcile these CRDs with cloud provider and on-prem APIs (e.g., GCP, AWS, or any API really)



XRDs

Platform API Schema

Crossplane

Define the shape of your API

```yaml
apiVersion: apiextensions.crossplane.io/v2
kind: CompositeResourceDefinition
metadata:
  name: databases.acme.com
spec:
  group: acme.com
  names:
    kind: Database
    plural: databases
  scope: Namespaced
  versions:
  - name: v1
    referenceable: true
    served: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            type: object
            properties:
```

Create XRD to declare our custom platform API

Custom API Group and Kind

Standard OpenAPI v3 Schema

Choose the config “knobs” to expose to devs

Composition

A function pipeline of business logic

Compositions

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: databases.acme.com
spec:
  compositeTypeRef:
    apiVersion: acme.com/v1
    kind: Database
  mode: Pipeline
  pipeline:
  - step: generate-resources
    functionRef:
      name: function-acme-func
    input: {}
  - step: filter-resources
    functionRef:
      name: function-filter
    input: {}
```

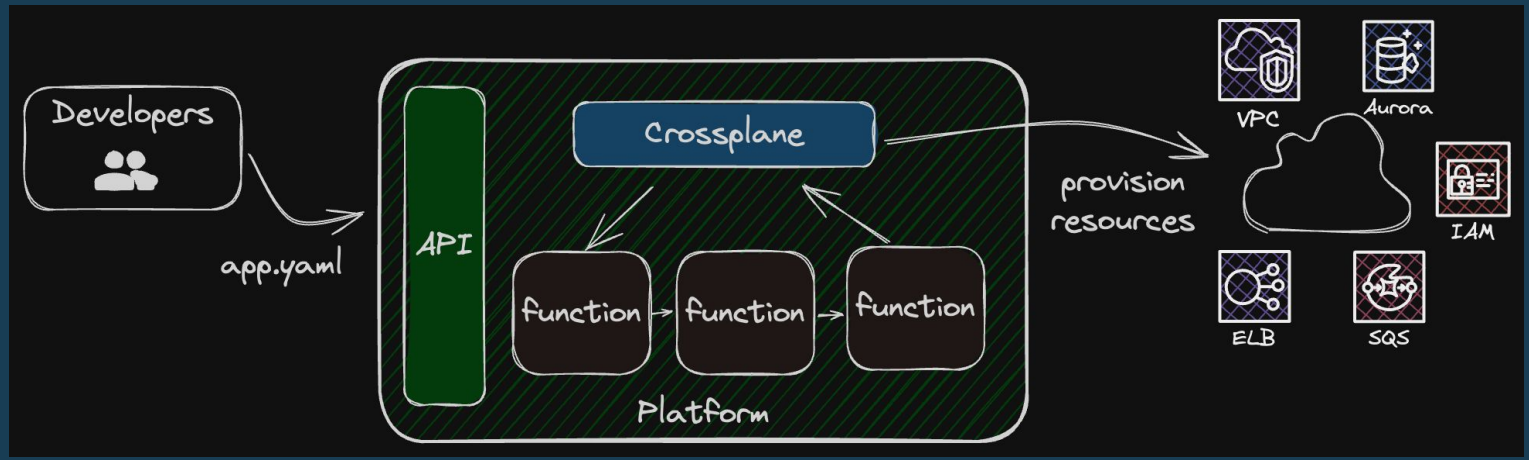
Define a Composition which implements the XRD

The XRD this Composition is for

Pipeline of functions to execute that will generate managed resources

How do Functions work?

- Compositions run a pipeline of simple functions to compose resources
- Written in the language/config of choice
- Focus only on your unique logic
- Crossplane does the heavy lifting of resources CRUD, reconciling, finalizers, owner refs, etc.



Templating

```
pipeline:
- step: render-templates
  functionRef:
    name: function-go-templating
  input:
    apiVersion: gotemplating.fn.crossplane.io/v1beta1
    kind: GoTemplate
    inline:
      template: |
        {{- range $i := until ( .observed.composite.resource.spec.count | int ) }}
        apiVersion: iam.aws.upbound.io/v1beta1
        kind: AccessKey
        spec:
          forProvider:
            userSelector:
              matchLabels:
                crossplane.io/name: user-{{ $i }}
        {{- end }}
```

Python

```
pipeline:
- step: compose-a-resource-with-python
  functionRef:
    name: function-python
  input:
    apiVersion: python.fn.crossplane.io/v1beta1
    kind: Script
    script: |
      from crossplane.function.proto.v1 import run_function_pb2 as fnv1

      def compose(req: fnv1.RunFunctionRequest, rsp: fnv1.RunFunctionResponse):
          rsp.desired.resources["bucket"].resource.update({
              "apiVersion": "s3.aws.upbound.io/v1beta2",
              "kind": "Bucket",
              "spec": {
                  "forProvider": {
                      "region": req.observed.composite.resource["spec"]["region"]
                  }
              },
          })
          rsp.desired.resources["bucket"].ready = True
```

KCL

```
pipeline:
- step: render-instances
  functionRef:
    name: kcl-function
  input:
    apiVersion: krm.kcl.dev/v1alpha1
    kind: KCLInput
    spec:
      source: |
        regions = ["us-east-1", "us-east-2"]
        items = [{
            apiVersion: "ec2.aws.upbound.io/v1beta1"
            kind: "Instance"
            metadata.name = "instance-" + r
            spec.forProvider: {
                ami: "ami-0d9858aa3c6322f73"
                instanceType: "t2.micro"
                region: r
            }
        } for r in regions]
```

Crossplane

Thanks @Peefy!

CUE

```
// if a base ARN exusts, render a policy with all ARNs.
if baseARN != "unknown" {
    let allTuples = list.Concat([
        [baseARN, baseARN + "/*"],
        [
            for a in additionalARNs {[a, a + "/*"]},
        ],
    ])
    let allResources = list.FlattenN( allTuples, 1)
    response: desired: resources: iam_policy: resource: {
        apiVersion: "iam.aws.upbound.io/v1beta1"
        kind:       "Policy"
        metadata: {
            name: "\(compName)-access-policy"
        }
        spec: {
            forProvider: {
                path: "/"
                policy: json.Marshal({
                    Version: "2012-10-17"
                    Statement: [
                        {
                            Sid: "S3BucketAccess"
                            Action: [
                                "s3:GetObject",
                                "s3:PutObject",
                            ]
                            Effect:   "Allow"
                            Resource: allResources
                        },
                    ]
                })
            }
        }
    }
}
```

Crossplane

Thanks @gotwarlost!

KRO

```yaml
pipeline:
- step: kro-run
  functionRef:
    name: function-kro
  input:
    apiVersion: kro.fn.crossplane.io/v1beta1
    kind: ResourceGraph
    resources:
    - id: vpc
      template:
        apiVersion: ec2.aws.m.upbound.io/v1beta1
        kind: VPC
        spec:
          forProvider:
            region: ${schema.spec.region}
            cidrBlock: 192.168.0.0/16
    - id: subnets
      forEach:
        - az: ${schema.spec.availabilityZones}
      template:
        apiVersion: ec2.aws.m.upbound.io/v1beta1
        kind: Subnet
        spec:
          forProvider:
            region: ${schema.spec.region}
            availabilityZone: ${schema.spec.region + az.suffix}
            cidrBlock: ${az.cidrBlock}
            vpcId: ${vpc.status.atProvider.id}
```

HCL

```
pipeline:
  - functionRef:
      name: fn-hcl
    step: run hcl composition
    input:
      apiVersion: function-hcl/v1
      kind: HclInput
      source: Inline
      hcl: |

        resource my-bucket {
          body = {
            apiVersion : "s3.aws.upbound.io/v1beta1"
            kind : "Bucket"
            metadata : {
              name : "${req.composite.metadata.name}-bucket"
            }
            spec : {
              forProvider : {
                region : req.composite.spec.parameters.region
              }
            }
          }
        }
```

Crossplane

Thanks @gotwarlost!

Full Code - General Purpose Programming

```
// RunFunction observes an example composite resource (XR). It simple adds one
// S3 bucket to the desired state.
func (f *Function) RunFunction(_ context.Context, req *fnv1beta1.RunFunctionRequest)
(*fnv1beta1.RunFunctionResponse, error) {
    f.log.Info("Running Function", "tag", req.GetMeta().GetTag())
    rsp := response.To(req, response.DefaultTTL)

    // create a single test S3 bucket
    _ = v1beta1.AddToScheme(composed.Scheme)
    name := "test-bucket"
    b := &v1beta1.Bucket{
        ObjectMeta: metav1.ObjectMeta{
            Annotations: map[string]string{
                "crossplane.io/external-name": name,
            },
        },
        Spec: v1beta1.BucketSpec{
            ForProvider: v1beta1.BucketParameters{
                Region: ptr.To[string]("us-east-2"),
            },
        },
    }

  ...

    return rsp, nil
}
```

Crossplane

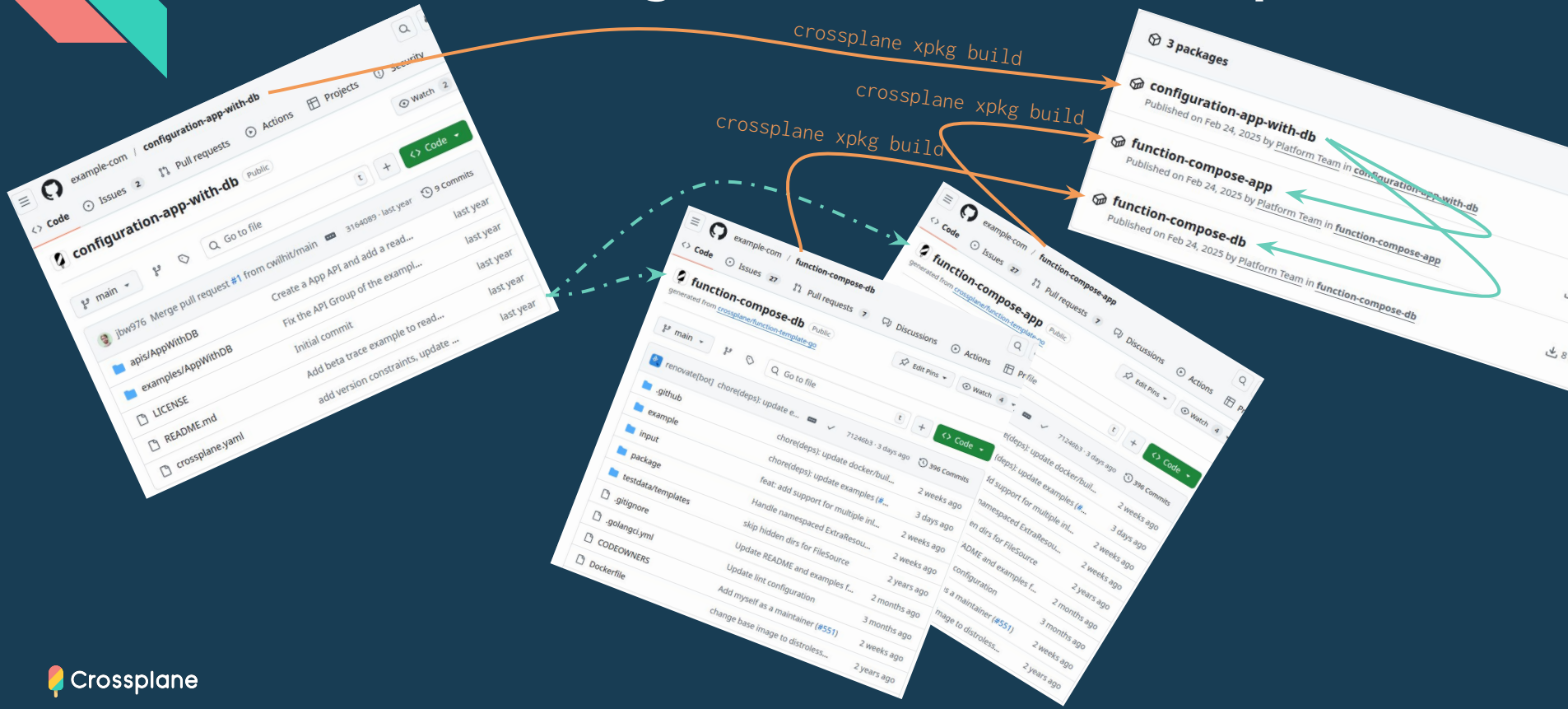
...or plain ol' YAML

```yaml
pipeline:
- step: patch-and-transform
  functionRef:
    name: function-patch-and-transform
  input:
    apiVersion: pt.fn.crossplane.io/v1beta1
    kind: Resources
    resources:
    - name: bucket
      base:
        apiVersion: s3.aws.upbound.io/v1beta1
        kind: Bucket
      patches:
      - type: FromCompositeFieldPath
        fromFieldPath: "spec.location"
        toFieldPath: "spec.forProvider.region"
        transforms:
        - type: map
          map:
            EU: "eu-north-1"
            US: "us-east-2"
```

Developer Experience

Simplifying Platform Building

Problem: Building with Functions is Complex



Crossplane

Problem: Embedding Code in YAML is Gross

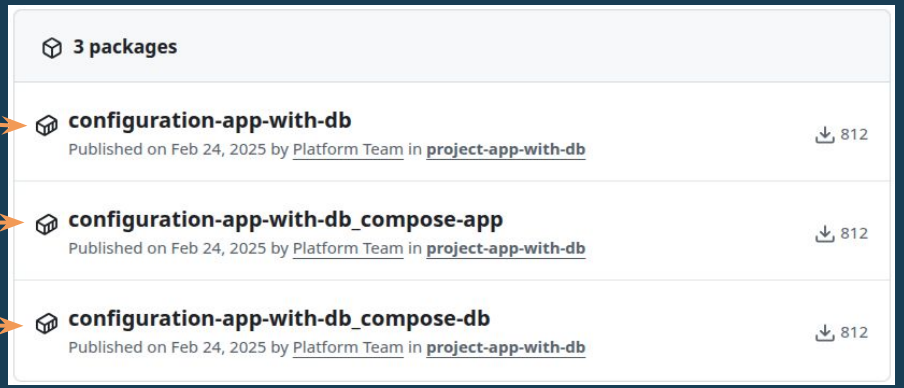
```yaml
apiVersion: apiextensions.crossplane.io/v1 all.json
kind: Composition
metadata:
  name: compose-a-resource-with-python
spec:
  compositeTypeRef:
    apiVersion: example.crossplane.io/v1
    kind: XR
  mode: Pipeline
  pipeline:
  - step: compose-a-resource-with-python
    functionRef:
      name: function-python
    input:
      apiVersion: python.fn.crossplane.io/v1beta1
      kind: Script
      script: |
        from crossplane.function.proto.v1 import run_function_pb2 as fnv1

        def compose(req: fnv1.RunFunctionRequest, rsp: fnv1.RunFunctionResponse):
            rsp.desired.resources["bucket"].resource.update({
                "apiVersion": "s3.aws.upbound.io/v1beta2",
                "kind": "Bucket",
                "spec": {
                    "forProvider": {
                        "region": req.observed.composite.resource["spec"]["region"]
                    }
                },
            })
            rsp.desired.resources["bucket"].ready = True
```

```yaml
apiVersion: apiextensions.crossplane.io/v1 all.json
kind: Composition
metadata:
  name: xmockdatabases.quickstart.crossplane.io
spec:
  compositeTypeRef:
    apiVersion: quickstart.crossplane.io/v1alpha1
    kind: XMockDatabase
  mode: Pipeline
  pipeline:
  - step: compose
    functionRef:
      name: crossplane-contrib-function-kcl
    input:
      apiVersion: krm.kcl.dev/v1alpha1
      kind: KCLRun
      metadata:
        name: compose-database
      spec:
        target: Resources
        params:
          name: "input-instance"
        source: |
          oxr = option("params").oxr
          items = [{
              apiVersion: "nop.crossplane.io/v1alpha1"
              kind: "NopResource"
              metadata.name = oxr.metadata.name
              spec.forProvider = {
                  conditionAfter = [{
                      conditionStatus: "True"
                      conditionType: "Ready"
                      time: "5s"
                  }]
                  fields = {
                      instanceClass: oxr.spec.parameters.size
                      region: oxr.spec.parameters.region
                      storageGb: oxr.spec.parameters.storage
                  }
              }
          }]
```

Crossplane

Solution: Control Plane Projects

```
[project-app-with-db]$ tree -L 2
.
├── apis
│   └── AppWithDB
├── crossplane-project.yaml
├── examples
│   └── app.yaml
├── functions
│   ├── compose-app
│   └── compose-db
├── LICENSE
├── README.md
└── tests
    ├── e2etest-app
    └── test-app
```



Demo - DevEx

adamwg/kubecon-eu-2026-devex-demo

Resource State Metrics

Every Crossplane metric you've ever wanted

A History of Crossplane Metrics

- Crossplane has complexity ➡️ “more metrics in Crossplane plz”
  - requests for metrics on claims, annotations, MRs, status, etc.
- What we’ve been able to tell you:
  - 15 GKE Clusters are unhealthy
- What you actually want to know:
  - Which specific clusters are failing?
  - Which teams own the failing resources?
  - How long has each resource been in a failed state?
  - What is the ready/synced status per team or namespace?

Crossplane Metrics - Now

- Hesitant to implement every request for 2 reasons
  - ❌ Updating core Crossplane code to add new metrics
  - ❌ Cardinality explosions
- Set out to design a flexible and powerful Crossplane metrics experience similar to kube-state-metrics: #6865
- Converged with upstream effort for resource-state-metrics
- **crossplane-contrib/resource-state-metrics**
  - Try it now and provide feedback!
  - Thank you @haarchri!

Demo - All the Metrics

jbw976/demo-xp-rsm

Community is everything

Contributors welcome!

- Getting started guide - make your first contribution!
  - contributing dir in `github/crossplane/crossplane`
- Code contributions in core and ecosystem
  - lots of functions and providers to help on!
- Docs contributions
  - share your expertise with others
- Good first issues, P0/P1 issues, roadmap
- Mentorship opportunities

Questions?

- github.com/crossplane/crossplane
- crossplane.io
- slack.crossplane.io

Join Crossplane Slack!

slack.crossplane.io