KCD Taipei
Building Internal Platforms with Crossplane

Clément Blaise - Platform Engineer

Crossplane

The Project

Crossplane

What is Crossplane?

- A cloud native **control plane framework**
  - Reduce the complexity of managing K8s controller
  - Provision/manage all kind of resources
- **Compose** those resources into high level **abstractions**
  - Give your developers self-service provisioning
- Kubernetes is a great control plane for containers
  - Crossplane teaches it how to manage **everything** else

How it got started?

- Started in 2018 by founders of the Rook project
- Inspiration by their Experience with Rook
  - Self-managing distributed storage systems
  - Orchestrating Ceph using Kubernetes
    - Recognized Kubernetes' potential to manage resources beyond the cluster
    - Leveraged Kubernetes' extensibility through Custom Resource Definitions (CRDs)
- Core Principles
  - Separation of concerns
  - Declarative configuration
  - Active Reconciliation
  - Dynamic provisioning of resources

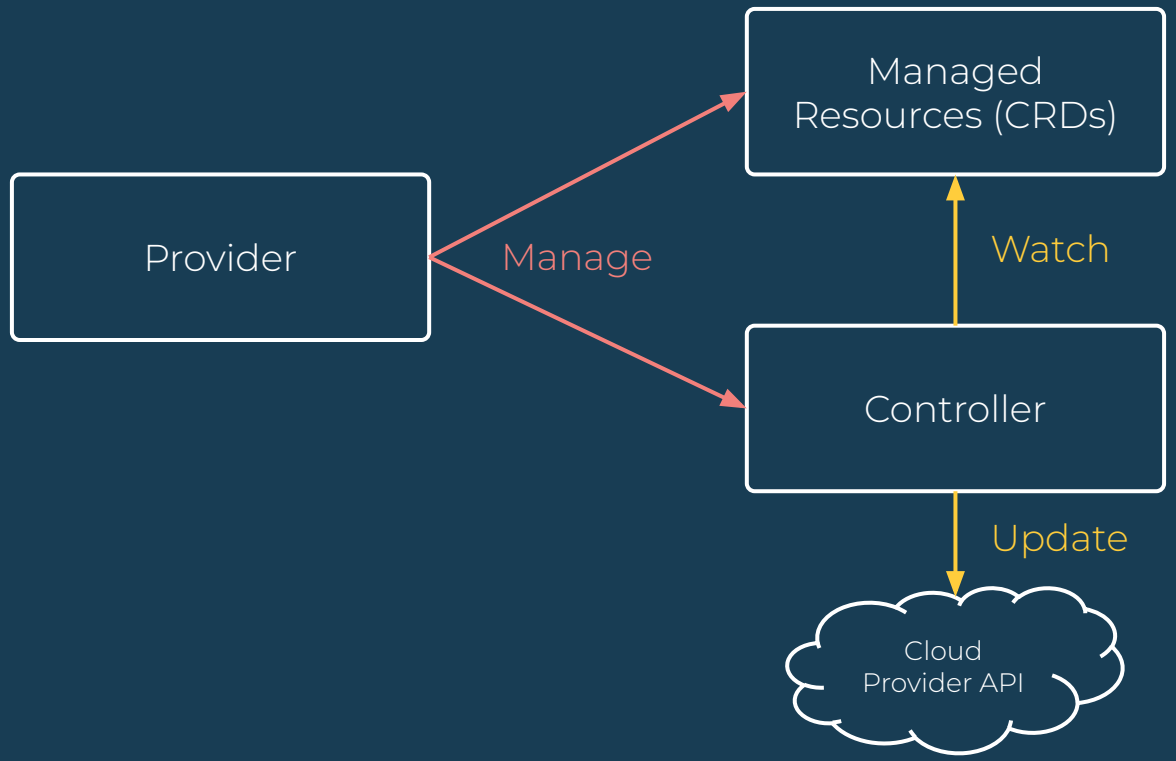
On the path to graduation

- Timeline:
  - CNCF Sandox in 2020
  - CNCF Incubating in 2021
  - CNCF Graduation proposal opened in 2024
- Lots of adopters in **production** and at **scale** (ADOPTERS.md)
  - Nike, Autodesk, Grafana, NASA Science Cloud, Elastic, Akamai, SAP, IBM, VMWare Tanzu, Nokia, etc.
- 2,325+ contributors to the project
- Steering committee has members from **Apple**, **Nokia**, and **Upbound** to lead and steward the project

The Basics

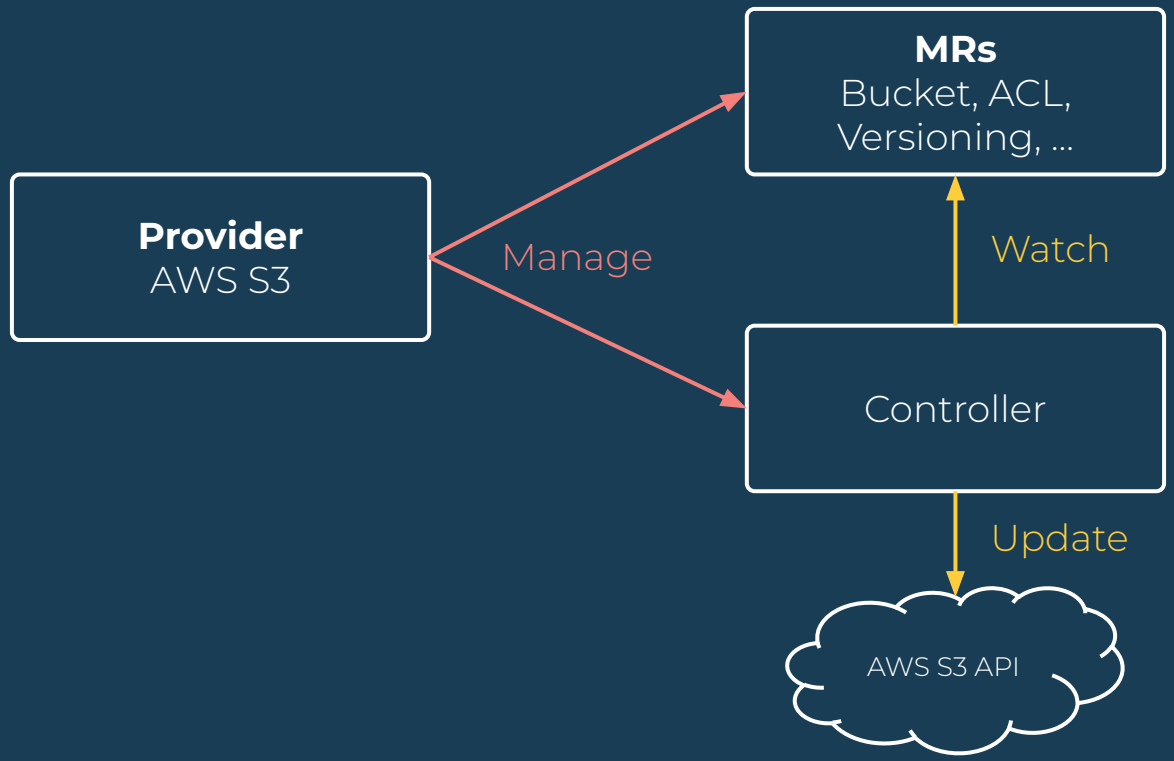
Providers

What is a Provider?



Crossplane

Example Provider: AWS S3



Crossplane

Managed Resources

```yaml
apiVersion: s3.aws.crossplane.io/v1beta1
kind: Bucket
metadata:
  name: crossplane-deepdive-demo-bucket
spec:
  forProvider:
    acl: private
    locationConstraint: eu-west-1
    paymentConfiguration:
      payer: BucketOwner
    versioningConfiguration:
      status: Enabled
    tagging:
      tagSet:
      - key: Name
        value: CrossplaneDeepDiveDemoBucket
```

Bucket overview

AWS Region	Amazon Resource Name (ARN)	Creation date
EU (Ireland) eu-west-1	arn:aws:s3:::crossplane-deepdive-demo-bucket	April 21, 2023, 15:00:23 (UTC+01:00)

Tags (1)

Track storage cost or other criteria by tagging your bucket. **Learn more**

Key	Value
Name	CrossplaneDeepDiveDemoBucket

Managed Resources

Status returned from the remote API

```
Status:
  At Provider:
    Arn:  arn:aws:s3:::crossplane-deepdive-demo-bucket

Events:
  Type    Age   From                        Message
  ----    ----  ----                        -------
  Normal  6m8s  bucket.s3.aws.crossplane.io  Successfully created external resource
```

Managed Resources Generate K8s Events

Crossplane

Building Providers and Evolutions

- Any language and tooling with a Kubernetes client
  - Golden Path
    - Golang: enable a shared language
    - Crossplane-runtime: set of libraries to interact with Crossplane
    - Kubebuilder: framework for building Custom Resource Definition
- Code Generation
  - Started with Cloud Provider Operators (ACK, ASO)
  - Terrajet: 1st generation of code generation from TF providers
  - Upjet: 2nd generation; customization points, generated documentation, reference inference and more stable

Upjet recent developments

- Support multi-version APIs
  - CRD versioning
  - Generate conversation webhook
- Managed resource metrics
  - Status
  - TTR (Time to Reconcile/Readiness)
  - Drift
- Support Terraform Plugin Framework
  - Avoid forking Terraform CLI

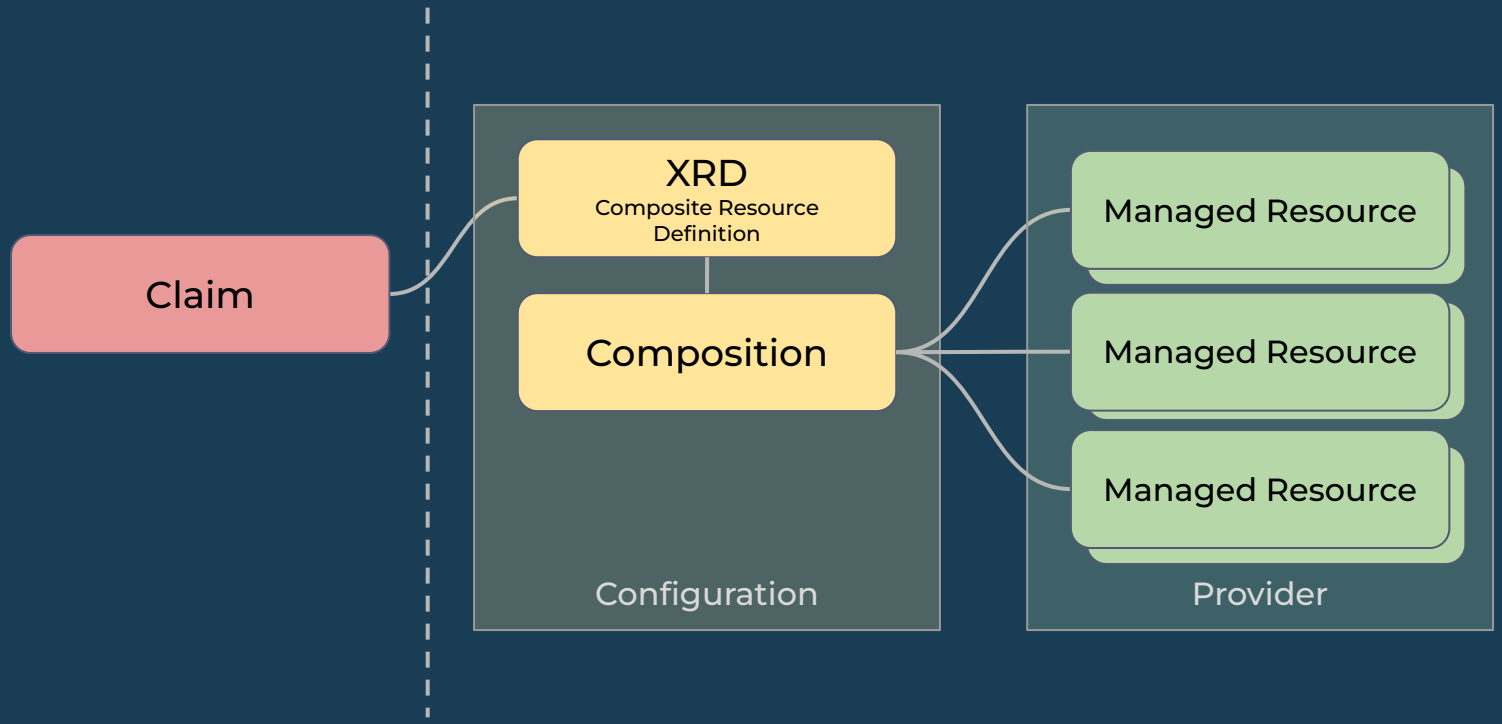
Building Your Control Plane

Composition and Functions

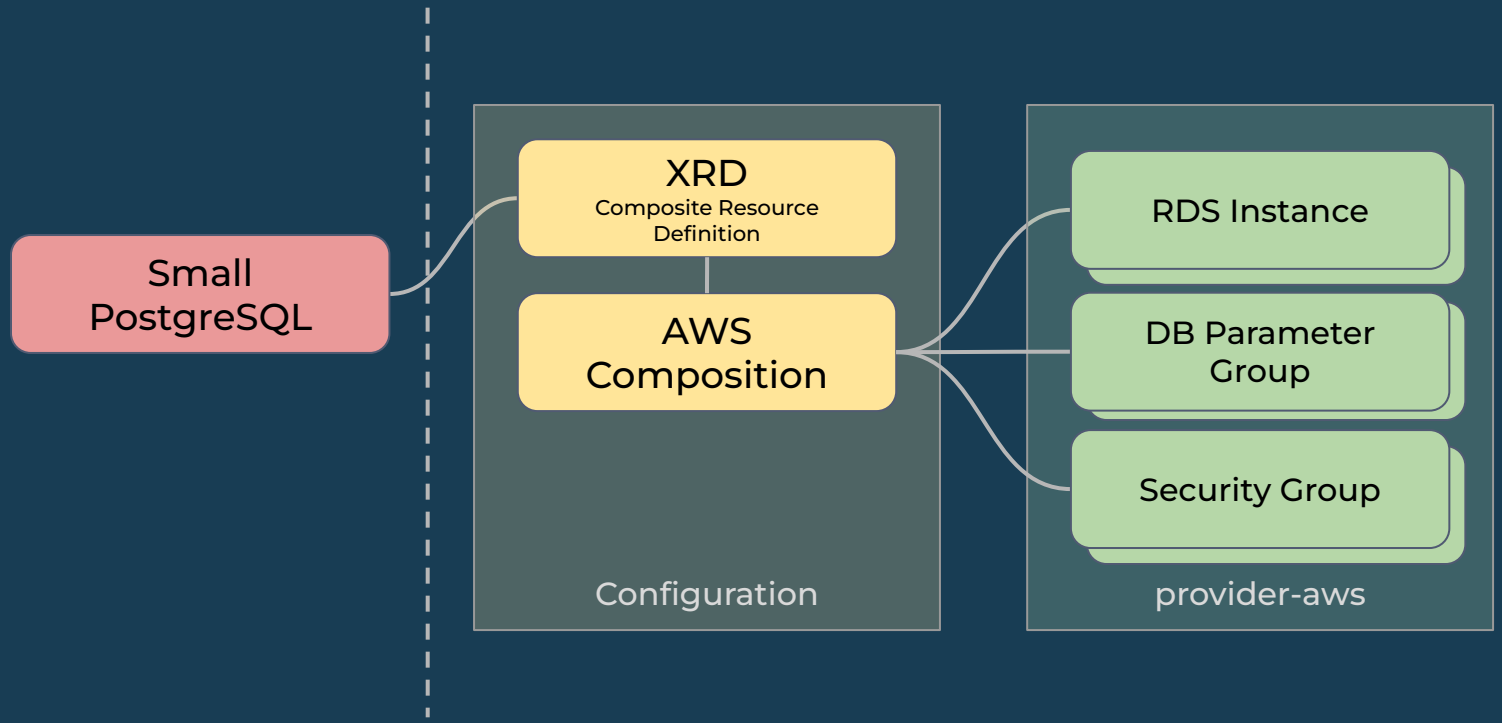
Crossplane

Build your own Platform API

- Assemble granular resources, e.g. from multiple clouds.
- Expose as higher level self-service API for your app teams
  - **Compose** EKS Cluster, VPC, Subnets, NAT GW, EIP
  - **Offer** as a single `Cluster` abstraction (API) with limited config for developers to self-service
- Hide infrastructure complexity and include policy guardrails
- All with K8s API - compatible with kubectl, GitOps, etc.
- No code **required**


Claim
XRD
Composite Resource Definition
Composition
Configuration
Managed Resource
Managed Resource
Managed Resource
Provider

Crossplane


Small PostgreSQL
XRD
Composite Resource Definition
AWS Composition
Configuration
RDS Instance
DB Parameter Group
Security Group
provider-aws

Crossplane

Composite Resource Definition

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: nosqls.database.example.com
spec:
  group: database.example.com
  names:
    kind: NoSQL
    plural: nosqls
  versions:
  - name: v1alpha1
    served: true
    referenceable: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
```

XRDs declare custom platform API

Custom API Group

Standard OpenAPIv3 Schema

Crossplane

Compositions

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
 name: nosqls.database.example.com
spec:
  compositeTypeRef:
    apiVersion: database.example.com/v1alpha1
    kind: NoSQL
  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

Crossplane

What can Functions do?

- Run a pipeline of functions to compose resources
- Written in your language of choice with your unique logic
- Sweet spot between “no code” vs building an entire controller
  - Focus only on your platform’s unique needs
  - Crossplane does the heavy lifting of CRUD-ing resources, reconciling, finalizers, owner refs, etc
- Writing code is **optional**
  - e.g., go templates, CUE scripts, KCL code, etc.
  - Reusable functions that are generally useful

Writing Functions - Go

```go
// 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

Using Functions - for loop (templates)

```yaml
apiVersion: apiextensions.crossplane.io/v1beta1
kind: Composition
metadata:
  name: example
spec:
  compositeTypeRef:
    apiVersion: database.example.org/v1
    kind: XPostgreSQLInstance
  mode: Pipeline
  pipeline:
  - step: compose-xr-using-go-templates
    functionRef:
      name: go-templates
    input:
      apiVersion: example.org/v1
      kind: GoTemplate
      source: Inline
      inline: |
        {{- range $i := until ( .desired.composite.resource.spec.count ) }}
        ---
        apiVersion:  rds.aws.upbound.io/v1beta1
        kind: Instance
        spec:
          forProvider:
            engine: postgres
            engineVersion: "13.7"
            ...etc...
        {{- end }}
```

Crossplane

Migrating to Functions - P&T

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: example
spec:
  compositeTypeRef:
    apiVersion: database.example.org/v1
    kind: XPostgresSQLInstance
  mode: Pipeline
  pipeline:
    - step: patch-and-transform
      functionRef:
        name: function-patch-and-transform
      input:
        apiVersion: pt.fn.crossplane.io/v1beta1
        kind: Resources
        resources:
          - name: database
            base:
              apiVersion: rds.aws.upbound.io/v1beta1
              kind: Instance
              spec:
                forProvider:
                  engine: postgres
                  engineVersion: "13.7"
```

Crossplane

The Kubernetes Controller Paradigm

Powering Control Planes

Crossplane

Control Plane Approach with Crossplane

- Unified Resource Management
  - Single API to manage infrastructure and application
  - Consistent interface for various services
- Declarative Configuration
- Multi-Cloud Compatibility
  - Abstract away provider-specific details
  - Promotes cloud-agnostic designs reducing vendor lock-in
- Extensibility

Advantages of Control Plane Approach

- Simplified Operations
  - Centralized management: reduces complexity and operational overhead
  - Increase consistency across environments and projects
- Developer Experience
  - Self-service provisioning
  - Faster onboarding
  - Reduce context-switching
- Consistent Governance
  - Enforce policies/compliance, easier auditing

K8s Controller Characteristics

- Resilient and Self-Healing
  - Continuous reconciliation loop
  - **Level-triggered** vs. edge-triggered
- Simplified State Management
  - Focus on describing desired end-state rather than transition logic
- Efficient Processing
  - Deduplication of items in the workqueue
  - No concurrent reconciliations
  - Rate-limited retries with exponential backoff
- Reliability
  - Reprocess all items at startup
  - Read K8s resources from an in-memory cache

Implementing Crossplane

A Step-by-Step Approach

Crossplane

Starting Small and Scaling Up

- Address Immediate Needs
  - Database Permissions Management
    - User creation and granting permissions
    - Automated secret generation for developer access
  - Service Account Automation
    - Example: IRSA for AWS resource interaction
- Scale to Cluster Provisioning
  - Automate whole cluster creation
  - Standardize across environments

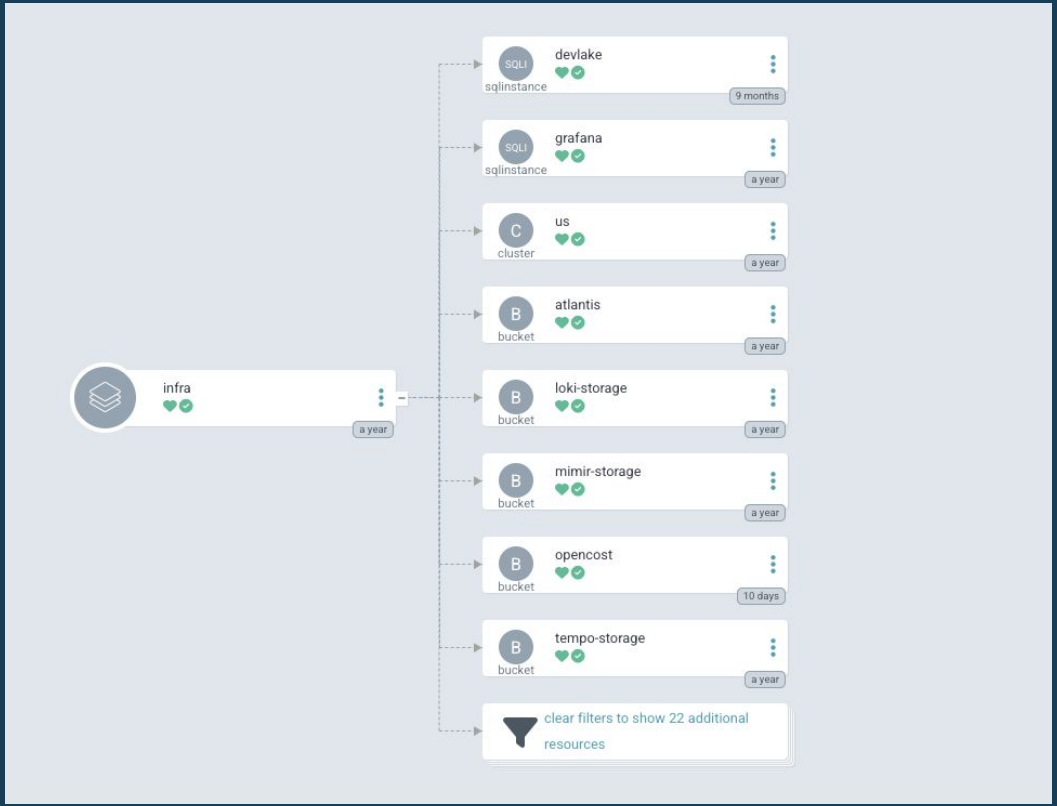
Expanding

- Integrate Cluster-Consumed Services
  - Databases
  - Caches
  - Container registries
  - Storage solutions
- Key Principles for Adoption
  - Implement gradually
  - Start where other tools have gaps
  - Build complexity as team familiarity grows
  - Continuously improve and expand your IDP

How to integrate your customs APIs?

GitOps

Argo CD and Crossplane Claims



Crossplane

Backstage Template

New Kubernetes Cluster

Request a Kubernetes Cluster to host workloads

Tenant | Environment | 3 Cluster Configuration | 4 NodeGroup Configuration | 5 GitOps | 6 Review

Name*
kcd
Name of this Cluster that other objects will use to refer to it
Cloud resources will be created as <environment>-<name>

Region*
Asia Pacific (Singapore)
Region is the region you'd like your resource to be created in.

Version
1.29
Kubernetes version of the Cluster

Network ID
1
Network ID for the VPC, a number between 0 and 255 templated as 10.networkId.0.0/16

Auto Scaler
Karpenter
Add autoscaling capability

Top Level Domain
kcd.demo
TLD is the DNS zone that will avaible for ingress resources

AddOn

- [ ] Stackrox
- [ ] Downscaler

Back | Next

Fill → Form

Templates resources
&
Open PR

Crossplane

Custom Integrations

POST, PUT, PATCH, DELETE, GET

/apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME

Get Involved

- Website: https://crossplane.io/
- Docs: https://crossplane.io/docs
- 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

Q&A