# Crossplane

13-September-2021
By Christian Boullosa

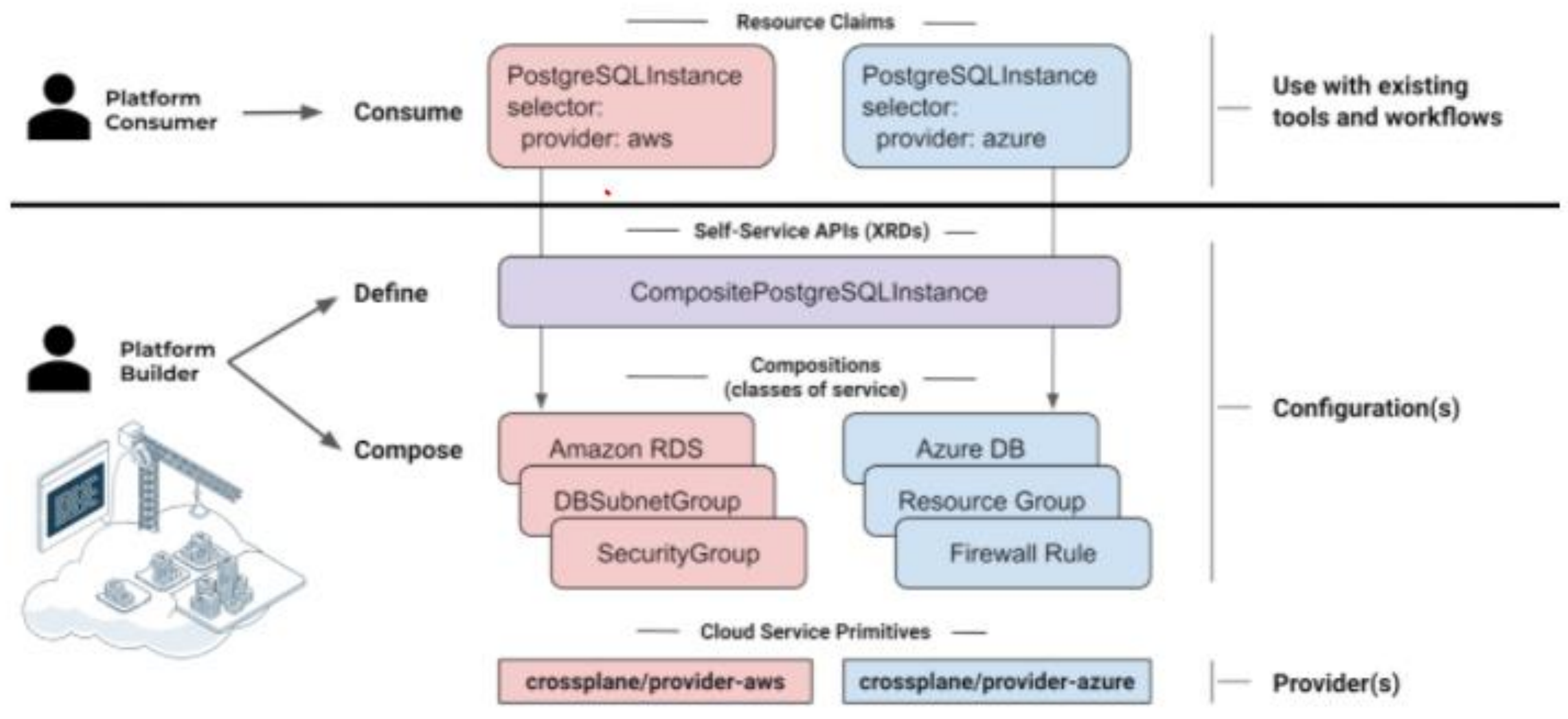
CLOUD NATIVE
COMPUTING FOUNDATION

## About Crossplane

https://crossplane.io/

- Crossplane is an open source Kubernetes add-on that enables platform teams to assemble infrastructure from multiple vendors, and expose higher level self-service APIs for application teams to consume, without having to write any code
- Launched in DEC 2018, by the creators of Rook(CNCF graduated project)
- Based on Kubernetes control plane
- Started by Upbound and adopted as a sandbox project by the cloud native community

# Multi-cloud Management Interface



# Features

Crossplane

- Provision and manage cloud infrastructure and services using kubectl
  - Crossplane extends your Kubernetes cluster, providing you with CRDs for any infrastructure or managed service
- There is a flavor of infrastructure for everyone on Crossplane
  - Crossplane supports infrastructure from all the major cloud providers and our community is constantly working on new Providers
- Publish simplified infrastructure abstractions for your applications
  - Build your own internal infrastructure abstractions on top of the CRDs Crossplane provides.
- The Universal Cloud API
  - Crossplane provides a consistent API across a diverse set of vendors, resources, and abstractions
- Run Crossplane anywhere
  - Whether you're using a single Kubernetes cluster in EKS, AKS, GKE, ACK, PKS or a multi-cluster manager like Rancher or Anthos, Crossplane integrates nicely with all of them

# Providers and Community

Crossplane

## Cloud Providers:

EQUINIX | METAL  aws  Alibaba Cloud  Google Cloud

Microsoft Azure  Red Hat

## Community:

upbound  Alibaba Group  Red Hat  GitLab  Microsoft

## Competitors

Crossplane

puppet labs

CHEF

SALTSTACK

ANSIBLE

HashiCorp Terraform

pulumi

Cloud Native Infrastructure as Code

# Crossplane  Pros & Cons

Pros:

- Kubernetes like/friendly setup (ymal, manifest, etc.)
- Good to manage drifting
- Common API with GitOps capabilities

Cons:

- Requires a Kubernetes cluster
- Not as many Cloud Providers as other solutions such as Terraform
- Poor Documentation

## Concepts: Providers

### Providers:

- Providers extend Crossplane to enable infrastructure resource provisioning. In order to provision a resource, a Custom Resource Definition (CRD) needs to be registered in your Kubernetes cluster and its controller should be watching the Custom Resources those CRDs define. Provider packages contain many Custom Resource Definitions and their controllers.

### Installing Providers:

- The core Crossplane controller can install provider controllers and CRDs for you through its own provider packaging mechanism, which is triggered by the application of a Provider resource

### Configuring Providers:

- In order to authenticate with the external provider API, the provider controllers need to have access to credentials. It could be an IAM User for AWS, a Service Account for GCP or a Service Principal for Azure. Every provider has a type called ProviderConfig that has information about how to authenticate to the provider API.

# Concepts: Providers Samples

1. Installing :

```
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: provider-aws
spec:
  package: "crossplane/provider-aws:master"
```

The field `spec.package` is where you refer to the container image of the provider. Crossplane Package Manager will unpack that container, register CRDs and set up necessary RBAC rules and then start the controllers.

There are a few other ways to trigger the installation of provider packages:
•As part of Crossplane Helm chart by adding the following statement to your `helm install` command: `--set provider.packages={crossplane/provider-aws:master}`.
•Using the Crossplane CLI: `kubectl crossplane install provider crossplane/provider-aws:master`

2. Configuring Providers:

```
apiVersion: aws.crossplane.io/v1beta1
kind: ProviderConfig
metadata:
  name: aws-provider
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: aws-creds
      key: key
```

You can see that there is a reference to a key in a specific `Secret`. The value of that key should contain the credentials that the controller will use. The documentation of each provider should give you an idea of how that credentials blob should look like. See Getting Started guide for more details.

https://crossplane.io/docs/v1.4/concepts/providers.html

# Concepts: Manage Resources

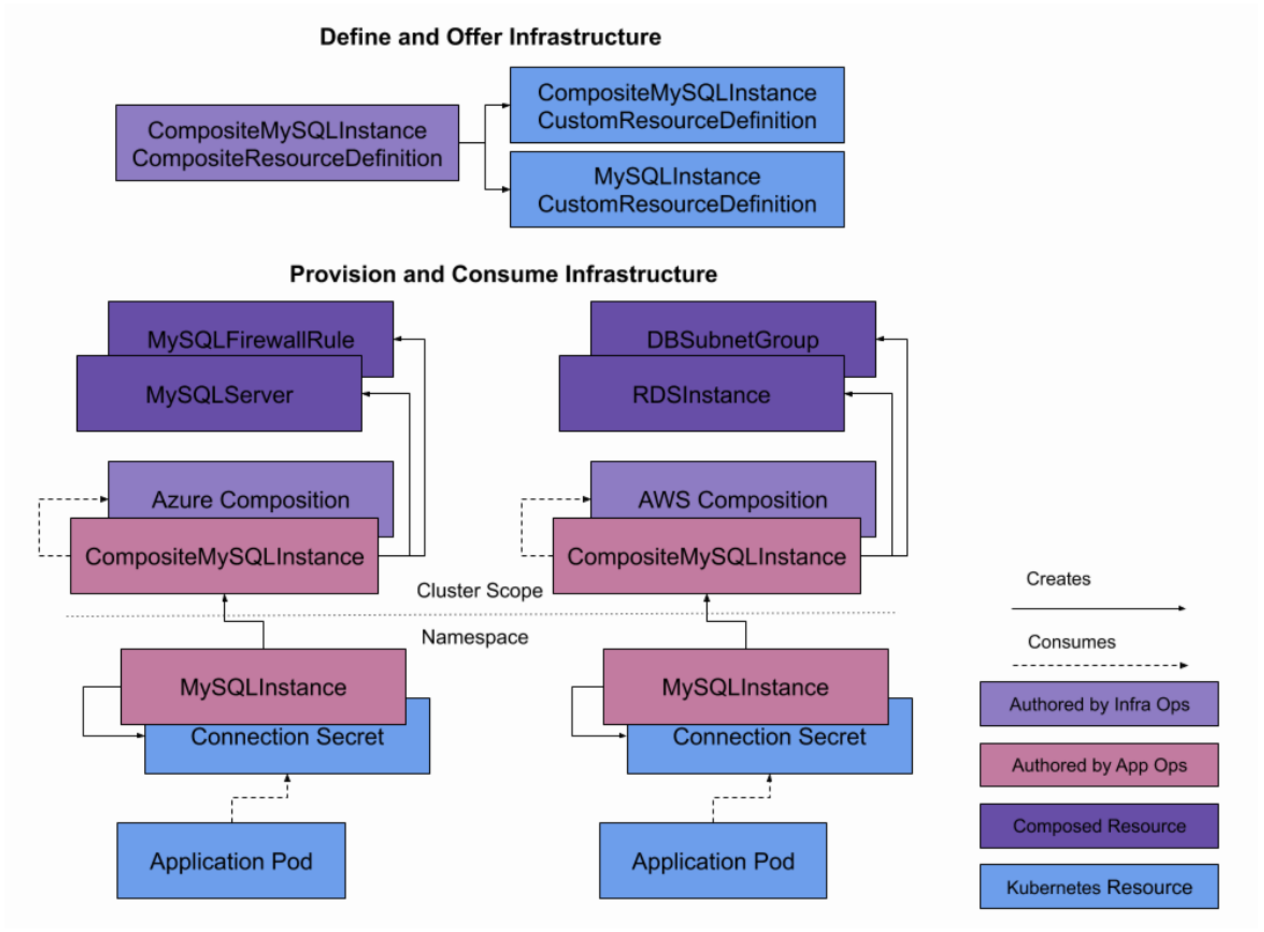
Managed resources are the Crossplane representation of the cloud provider resources, and they are considered primitive low level custom resources that can be used directly to provision external cloud resources for an application or as part of an infrastructure composition.

- Syntax: Crossplane API conventions extend the Kubernetes API conventions for the schema of Crossplane managed resources. Following is an example of RDSInstance:

```yaml
apiVersion: database.aws.crossplane.io/v1beta1
kind: RDSInstance
metadata:
  name: foodb
spec:
  forProvider:
    dbInstanceClass: db.t2.small
    masterUsername: root
    allocatedStorage: 20
    engine: mysql
  writeConnectionSecretToRef:
      name: mysql-secret
      namespace: crossplane-system
  providerConfigRef:
    name: default
  deletionPolicy: Delete
```

# Concepts: Composing Infrastructure

Composition allows platform builders to define new custom resources that are composed of managed resources. We call these composite resources, or XRs. An XR typically groups together a handful of managed resources into one logical resource, exposing only the settings that the platform builder deems useful and deferring the rest to an API-server-side template we call a 'Composition'.



## Concepts: Composing Infrastructure

Crossplane

- A **Composite Resource (XR)** is a special kind of custom resource that is composed of other resources. Its schema is user-defined. The CompositeMySQLInstance in the above diagram is a composite resource. The kind of a composite resource is configurable - the Composite prefix is not required.
- A **Composition** specifies how Crossplane should reconcile a composite infrastructure resource - i.e. what infrastructure resources it should compose. For example, the Azure Composition configures Crossplane to reconcile a CompositeMySQLInstance by creating and managing the lifecycle of an Azure MySQLServer and MySQLServerFirewallRule.
- A **Composite Resource Claim (XRC**) for a resource declares that an application requires particular kind of infrastructure, as well as specifying how to configure it. The MySQLInstance resources in the above diagram declare that the application pods each require a CompositeMySQLInstance. As with composite resources, the kind of the claim is configurable. Offering a claim is optional.
- A **CompositeResourceDefinition (XRD)** defines a new kind of composite resource, and optionally the claim it offers. The CompositeResourceDefinition in the above diagram defines the CompositeMySQLInstance composite resource, and its corresponding MySQLInstance claim

# Concepts: Packages

Crossplane packages are opinionated OCI images that contain a stream of YAML that can be parsed by the Crossplane package manager. Crossplane packages come in two varieties: Providers and Configurations. Ultimately, the primary purposes of Crossplane packages are as follows:

- **Convenient Distribution**: Crossplane packages can be pushed to or installed from any OCI-compatible registry.
- **Version Upgrade**: Crossplane can update packages in-place, meaning that you can pick up support for new resource types or controller bug-fixes without modifying your existing infrastructure.
- **Permissions**: Crossplane allocates permissions to packaged controllers in a manner that ensures they will not maliciously take over control of existing resources owned by other packages. Installing CRDs via packages also allows Crossplane itself to manage those resources, allowing for powerful composition features to be enabled.
- **Dependency Management**: Crossplane resolves dependencies between packages, automatically installing a package’s dependencies if they are not present in the cluster and checking if dependency versions are valid if they are already installed.

# Installation Notes

1. Choosing Hosted or Self-Hosted Crossplane
   - Hosted: Using Upbound Cloud: https://upbound.io/
   - Self-Hosted: or Using your own Kubernetes cluster ⟸

Requires a k8s Cluster, i.e you can use minikube

2. You Still need to Install CLI
3. Install and Configure
   - Select a Getting Started Configuration
   - Install Configuration Package
   - Account Key (depending on Provider)
   - Create a Provider Secret
   - Configure Provider
4. Provision Configuration
   - Claim your infrastructure
   - Consume  your Infrastructure

## Demo

Crossplane

1. Installing and configuring Crossplane inside GCP cluster
2. Provision Infrastructure per Crossplane pre-made packages, Crossplane yaml manifest
   - https://raw.githubusercontent.com/crossplane/crossplane/release-1.4/docs/snippets/compose/claim-gcp.yaml
3. Create another cluster using Crossplane
4. Test Drifting scenario:
   - Claim your infrastructure Modifying cluster node zones manually via GCP console
   - Monitor the change and see how Crossplane brings the zones back to normal
   - Push the wanted changes with github and see Crossplane in action

- **Assumption:** Argos and Helm must be installed and configured already in your cluster

## End

Questions?

CLOUD NATIVE
COMPUTING FOUNDATION