International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

# Managing Cloud Infrastructure the Kubernetes Way: A Practical Take on Crossplane's Declarative Power

**Sijo Joseph Nellissery**

Principal Engineer, SAP US, Palo Alto, CA

**Abstract:** ***Crossplane is a powerful, open-source control plane for managing infrastructure using Kubernetes-native APIs. It enables organizations to provision, manage, and compose cloud infrastructure directly from Kubernetes, making it easier for platform teams to expose infrastructure abstractions to developers. Unlike Terraform or Pulumi, Crossplane operates declaratively within the Kubernetes ecosystem, offering real-time reconciliation and continuous drift detection. This white paper outlines Crossplane's architecture, benefits, and a practical example of provisioning Google Cloud Platform (GCP) resources such as CloudSQL and GKE using Crossplane.***

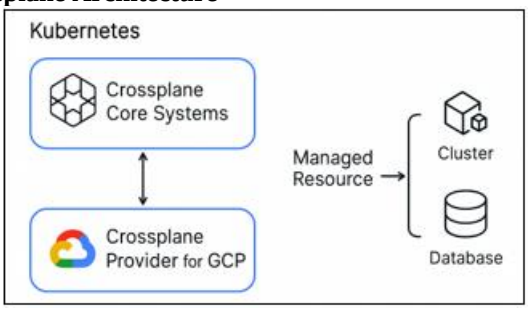
**Keywords:** Crossplane, Kubernetes-native infrastructure, declarative provisioning, cloud resource management, GCP automation

## 1. Introduction to Crossplane

As cloud adoption accelerates, enterprises increasingly face challenges in managing infrastructure across multiple cloud providers in a secure, scalable, and consistent manner. Crossplane is a powerful open-source control plane that brings Kubernetes-style declarative infrastructure management to cloud and on-prem environments. By extending Kubernetes with custom resource definitions (CRDs), Crossplane enables platform teams to define, compose, and publish infrastructure abstractions that can be consumed by application teams—without requiring them to write Terraform, YAML, or scripts.

Crossplane decouples infrastructure provisioning from cloud-specific APIs and embeds policy-driven controls, making it ideal for platform engineering, GitOps workflows, and multi-cloud governance. With support for major cloud providers like AWS, GCP, Azure, and Alibaba Cloud, Crossplane empowers teams to manage infrastructure as code, using Kubernetes-native tools, while adhering to organizational security and compliance standards.

**Crossplane Architecture**




Crossplane consists of the following key components:

**CRDs**: Crossplane CRDs (Custom Resource Definitions) are Kubernetes extensions that define the APIs used by Crossplane to provision and manage infrastructure resources in a cloud-native, declarative way.

In Crossplane, CRDs represent:

1) **Infrastructure Resources** – These are cloud resources like databases, buckets, virtual machines, and networks.
2) **Composite Resources (XRs)** – High-level abstractions composed of multiple infrastructure resources (e.g., a "KubernetesCluster" composed of a GKE cluster, VPC, and service account).
3) **Claims** – Application-facing representations of composite resources, often namespace-scoped and simplified for developer use.
4) **Configuration CRDs** – Allow platform teams to define reusable infrastructure blueprints, policies, and constraints.

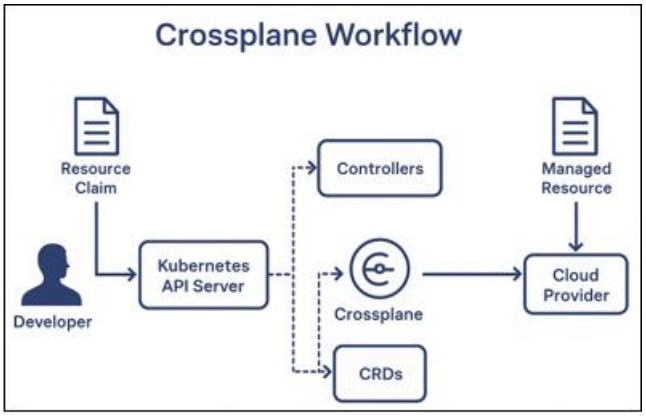
**Controllers**: In Crossplane, **controllers** are the core components responsible for reconciling custom resources (CRDs**)**—meaning they continuously ensure that the actual state of infrastructure matches the desired state declared in Kubernetes manifests.

A Crossplane controller is a Kubernetes controller that:

- **Watches** for changes to specific CRDs (e.g., Bucket, CloudSQLInstance, CompositeResource).
- **Reconciles** by making API calls to cloud providers to create, update, or delete real infrastructure to match the CRD spec.
- **Owns the control loop**: The controller constantly checks that the current state in the cloud matches the desired state described in the custom resource.

**Provider Configs**: ProviderConfig tells Crossplane how to authenticate with the external system and provides configuration options for how that connection should behave. Every time you create a managed resource (e.g., a Bucket, CloudSQLInstance, or VPC), Crossplane refers to the providerConfigRef field in that resource to determine which credentials/config to use for provisioning.

General workflow can be visualized as below




## 2. Key Features and Benefits

Crossplane is a Kubernetes-native control plane that enables platform teams to build and operate internal cloud platforms declaratively. It brings infrastructure-as-code concepts directly into Kubernetes using Custom Resource Definitions (CRDs) and controllers, extending Kubernetes from application deployment to full-stack cloud infrastructure provisioning.

Kubernetes-Native Infrastructure Management
Crossplane allows you to define, provision, and manage infrastructure resources—like databases, networks, and compute instances—as Kubernetes-native custom resources. This means your entire platform, from applications to infrastructure, can be managed through a unified Kubernetes API.

Separation of Concerns with Composition
Crossplane introduces a powerful abstraction called Composition, which lets platform teams define reusable templates for infrastructure (e.g., a "production database"), while application teams consume these templates as simple custom resources (e.g., CompositePostgreSQLInstance).

Support for Multi-Cloud and Hybrid Environments
Crossplane supports major cloud providers including AWS, GCP, Azure, and Alibaba Cloud via provider packages. You can switch cloud providers or even deploy across multiple clouds by just changing the ProviderConfig.

Dynamic, GitOps-Ready Infrastructure
Since everything in Crossplane is represented as declarative YAML, it integrates seamlessly with GitOps workflows using tools like ArgoCD and Flux. This allows teams to version-control and audit infrastructure changes alongside application deployments.

Secure, Scalable Credential Management
With ProviderConfigs, Crossplane securely manages cloud credentials. It supports multiple identity sources, including Kubernetes secrets, workload identity (GCP), IRSA (AWS), and more.

Composable Control Planes for Platform Engineering
Crossplane empowers platform engineers to build custom control planes tailored to their organization. These control planes abstract complexity, enforce compliance, and streamline provisioning via APIs customized to your team's needs.

Extensible and Open Ecosystem
Crossplane is extensible through providers and compositions and integrates with the CNCF ecosystem. It allows easy integration with secret managers, policy engines (like OPA/Gatekeeper), service meshes, and CI/CD systems.

**Crossplane Installation Guide with GCP Provider**
**Step 1: Add the Crossplane Helm Repository**

```
helm repo add crossplane-stable
https://charts.crossplane.io/stable

helm repo update
```

**Step 2: Install Crossplane into the crossplane-system Namespace**

```
kubectl create namespace crossplane-system

helm install crossplane \

  --namespace crossplane-system \

  crossplane-stable/crossplane
```

**Step 3: Verify Crossplane Installation**

```
kubectl get pods -n crossplane-system
```

Expected output includes:
- crossplane
- crossplane-rbac-manager

**Step 4: Install a Provider (e.g., GCP)**

```
kubectl crossplane install provider
crossplane/provider-gcp

kubectl get providers
```

**Step 5: Create a ProviderConfig**
You'll need to create a Kubernetes Secret with your cloud provider credentials (example for GCP):

```
kubectl create secret generic gcp-creds \
  --from-file=creds=./gcp-creds.json \
  -n crossplane-system
```

Then apply a ProviderConfig:

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

**GCP Resource Provisioning Example**
**CloudSQL Instance**

```
apiVersion:
database.gcp.crossplane.io/v1beta1
kind: CloudSQLInstance
metadata:
  name: sample-db
spec:
  forProvider:
```

```
    region: us-central1
    databaseVersion: POSTGRES_13
    settings:
      tier: db-f1-micro
  writeConnectionSecretToRef:
    name: db-conn
    namespace: default
  providerConfigRef:
    name: default
```

**GKE Cluster**

```
apiVersion:
container.gcp.crossplane.io/v1beta1
kind: GKECluster
metadata:
  name: demo-cluster
spec:
  forProvider:
    location: us-central1
    initialClusterVersion: "1.29"
    nodePools:
      - name: default-pool
        initialNodeCount: 2
        config:
          machineType: e2-medium
  writeConnectionSecretToRef:
    name: cluster-creds
    namespace: default
  providerConfigRef:
    name: default
```

**Compositions and Claims**

Platform teams can define a Composition to abstract complex infrastructure:

```
apiVersion:
apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: composite-postgres
spec:
  compositeTypeRef:
    apiVersion:
database.example.org/v1alpha1
    kind: XPostgresInstance
  resources:
    - name: cloudsql
      base:
        apiVersion:
database.gcp.crossplane.io/v1beta1
        kind: CloudSQLInstance
        spec:
          forProvider:
            databaseVersion: POSTGRES_13
            settings:
              tier: db-f1-micro
          providerConfigRef:
            name: default
```

Developers then create simple CompositeResourceClaims:

```
apiVersion:
database.example.org/v1alpha1
kind: PostgresInstanceClaim
metadata:
  name: dev-db
spec:
  compositionSelector:
    matchLabels:
      environment: dev
```

**Security and Access Control**

Crossplane's architecture is built with enterprise-grade security principles that ensure safe, compliant, and role-scoped infrastructure management directly from Kubernetes. By adopting Kubernetes-native mechanisms and extending them with provider-specific controls, Crossplane provides robust security and access control across environments.

Separation of Duties via Kubernetes RBAC:
Crossplane leverages Kubernetes Role-Based Access Control (RBAC) to enforce least-privilege access**.** Platform Engineers define and publish CompositeResourceDefinitions (XRDs) and Compositions. Application Developers consume Composite Resources (XRs) without access to cloud credentials or low-level provider configuration. This allows organizations to isolate who can define infrastructure versus who can consume it—mitigating the risk of misconfiguration or unauthorized access.

Scoped ProviderConfigs for Credential Isolation
Crossplane uses ProviderConfig objects to connect to external cloud providers. Each ProviderConfig references a Kubernetes Secret, typically storing cloud credentials. Security features include:

- Namespace Isolation**:** Credentials are stored in a dedicated namespace (crossplane-system), inaccessible to users without explicit permission.
- Multiple ProviderConfigs**:** Enables safe multi-tenancy each team or environment can use separate credentials or IAM scopes.
- Read-only Permissions**:** You can scope permissions (e.g., read-only vs. full access) per ProviderConfig using cloud IAM roles.

Write-Only Access to Secrets
Crossplane creates cloud resource connection secrets (e.g., database credentials, kubeconfigs) in target namespaces. You can restrict: Who can read these secrets (typically app teams), Who can write them (Crossplane controllers only), TTL or rotation policies (managed via external vault systems)

Secure Communication and Controller Behavior
Crossplane controllers operate inside the cluster, avoiding external calls unless provisioning is required. Communication with cloud providers happens through the provider's API using the credentials stored in ProviderConfig. You can enforce audit logging**,** rate limits**,** and network policies around controller pods.

GitOps and Secret Management Compatibility
Crossplane integrates seamlessly with GitOps tools like ArgoCD or Flux, and secrets can be managed via external tools (e.g., Vault**,** Sealed Secrets**,** External Secrets Operator) for additional compliance and automation.

## 3. Conclusion

Crossplane enables cloud infrastructure management using native Kubernetes tools and workflows. With its powerful composition and reconciliation capabilities, organizations can

build secure, scalable, and developer-friendly platforms while minimizing operational overhead.

By using Crossplane to manage GCP (or any major cloud) resources, enterprises can modernize their infrastructure practices and accelerate delivery without compromising on control or compliance.

## References

[1] Crossplane Official Documentation: https://docs.crossplane.io/ (Primary source for installation, CRDs, compositions, providers, and architecture)
[2] Crossplane GitHub Repository: https://github.com/crossplane/crossplane (Open-source codebase, controller definitions, releases, and community resources)
[3] Crossplane Blog & CNCF Case Studies: https://blog.crossplane.io/ https://www.cncf.io/blog/2022/02/09/crossplane-production-stories-from-the-community/ (Use cases, production success stories, ecosystem updates)
[4] Upbound Crossplane Marketplace & Providers: https://marketplace.upbound.io/providers (Cloud provider integrations and available managed resources)
[5] CNCF Landscape & Crossplane Profile: https://landscape.cncf.io/card-mode?project=crossplane (CNCF ecosystem overview and governance)
[6] Security Best Practices in Kubernetes: https://kubernetes.io/docs/concepts/security/ (Important context for secure Crossplane deployments)
[7] Crossplane Provider GCP Documentation: https://marketplace.upbound.io/providers/upbound/provider-gcp (Resource schema and example configurations for Google Cloud resources)