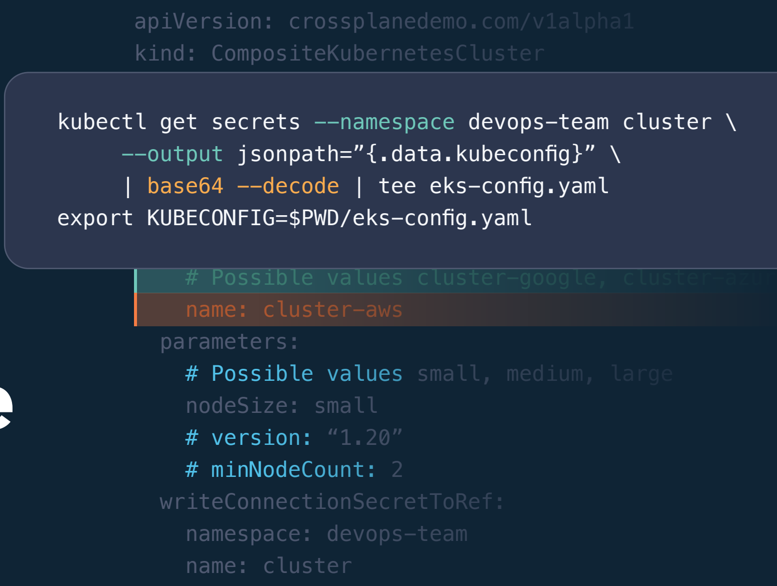
upbound



```
apiVersion: crossplanedemo.com/v1alpha1
kind: CompositeKubernetesCluster
```

```
kubectl get secrets --namespace devops-team cluster \
     --output jsonpath="{.data.kubeconfig}" \
     | base64 --decode | tee eks-config.yaml
export KUBECONFIG=$PWD/eks-config.yaml
```

```
    # Possible values cluster-google, cluster-
    name: cluster-aws
  parameters:
    # Possible values small, medium, large
    nodeSize: small
    # version: "1.20"
    # minNodeCount: 2
  writeConnectionSecretToRef:
    namespace: devops-team
    name: cluster
```

WHITEPAPER

Scaling Crossplane to Production

What is Crossplane?

Crossplane is a CNCF project that transforms your Kubernetes clusters into universal control planes. Crossplane enables platform teams to use Providers to install Custom Resource Definitions (CRDs), extending the Kubernetes API to provision, compose, and consume infrastructure from multiple vendors, without writing any code.

The building blocks of Crossplane are Providers and Managed Resources.

Providers tell Crossplane how to communicate to API endpoints external to Kubernetes. Though it is the most common use case, Providers aren't restricted to managing cloud resources; they can be written for anything that has an API - for example, to use Crossplane to order Dominos Pizza. The provider provides the rules that teach Crossplane how to structure API calls, authenticate, and what API endpoints exist.

Crossplane Managed Resources (MRs) are a kind of Kubernetes Custom Resource. The API server uses CRDs to learn about new kinds of Custom Resources; CRDs include all the information the API server needs to expose a new Custom Resource - for example: its type and OpenAPI schema. Depending on the system it will be communicating with, a single Provider might contain hundreds of MRs - AWS alone has almost 1000 MR API endpoints, nearly 20 times more than Kubernetes itself! Every external API needs an associated MR; for example, in the Crossplane AWS Provider, RDSInstance corresponds to an actual RDS Instance in AWS. The more services a Provider has, the more MRs need to be installed into Kubernetes.

What is Upbound?

Upbound is the company that invented the open source Crossplane project. Our mission is to help customers build Internal Cloud Platforms using control planes. Every Upbound customer gets access to Upbound Universal Crossplane (UXP) and Official Providers so they can build their Internal Cloud Platforms with Crossplane confidently. UXP is Upbound's downstream distribution of Crossplane, and Official Providers are production ready versions of Providers. Both are maintained, tested, and supported by Upbound on behalf of our customers and community.

Our mission is to help customers build Internal Cloud Platforms using control planes.

SOME BACKGROUND

There were only a few API endpoints in the original Kubernetes design. In 2019 when Kubernetes v1.16 made CRDs Generally Available, the idea of hundreds or thousands of CRDs was unheard of. With CRDs able to connect Kubernetes to anything, the Crossplane community saw an opportunity to make Kubernetes the one tool to build and manage infrastructure. To ensure high fidelity with, and full coverage of, cloud vendor APIs, Crossplane incorporated hundreds of CRDs per Provider. Need an AWS IAM Group? Crossplane has it. What about a GCP Autoscaler? Crossplane has that too. When users began managing multiple clouds with Crossplane, the number of CRDs installed to Kubernetes grew to thousands and tens of thousands.

SO WHAT IS THE PROBLEM?

The problem is Kubernetes doesn't scale well after you install a few hundred of these CRDs; the Kubernetes control plane's memory usage scales linearly with the number of CRDs installed. This can cause performance degradation and unresponsiveness. A typical Crossplane production deployment of 2,000-3,000 CRDs can be 2-3 times the number of CRDs present in that AWS Provider. What happens if you need to install all of the "Big 3" (AWS, GCP, Azure)?

So why does Kuberentes have this problem? Unfortunately, the original Kubernetes API implementation did a poor job of using system memory. Before Crossplane, when the number of API endpoints was low (~25-50), no one noticed a problem. For Crossplane users though, this became a major issue. Users reported unstable API Servers, Kubernetes commands that would take minutes to return a result or maybe never finish. Entire clusters became unusable, not just for CRDs but for everything else as well.

In order to support those thousands of CRDs introduced per Provider, Upbound looked at the entire Kubernetes stack from the Client request to the API Server processing and CRD data structures, resulting in multiple potential places for improvement across both clients and servers.

CLIENT-SIDE ISSUES

Kubernetes must periodically check for new CRDs you may have added to the cluster. This "api discovery" process can take a while because clients like kubectl and helm are configured to cache and rate limit queries - ironically, for performance reasons. More CRDs installed means more queries. Upbound, the creator of Crossplane and the industry's first platform for building internal cloud platforms, maintains a set of mature, feature-complete, enterprise-level "Official" Crossplane Providers. With Upbound's Official Providers, you get stable, full coverage parity with cloud vendor APIs. All resource APIs are production ready, giving platform teams a stable foundation to build their Internal Cloud Platforms.

Upbound's Marketplace has the perfect solutions to build internal cloud platforms. This is where to go to find Official Providers and featured configurations - custom blueprints and reference platforms to power your internal cloud platform. For Upbound's Official Providers, there are over 300 API groups. Because the default api rate limit is 50 queries per second with a burst of 100 queries per second, if there are more than 100 group versions, we will hit the rate limit during api discovery, which will slow down the process. We have seen delays of up to 10 minutes here.

10min

Delays caused by default API rate limits

300

Increased burst of queries per second

10x

Increased speed of an average kubectl queary against 300 API groups

CLIENT-SIDE SOLUTIONS

While there are other fixes coming, bumping this burst from 100 to 300, and flushing the discovery cache less frequently (every 6 hours instead of every 10 minutes) has already fixed these issues for most folks on the latest versions of kubectl. I.e., we can ask the server to do more work before it rate limits us, and faster: an average kubectl query against 300 API groups now takes one second (a 10x speedup!). The fix was introduced in kubectl and Kubernetes v1.25.0, and backported to Kubernetes v1.22.13, v1.23.20, v1.24.4; as long as you are using one of those versions, you will automatically take advantage of these improvements.

But why take your chances? Our advice is to keep up to date, or tell your cloud vendors to keep up to date! At the time of writing, AKS and GKE are 1 patch behind (and GKE's version requires a 20-node cluster to keep up), whereas EKS is 13 patches behind!

This is where Upbound Universal Crossplane comes in. As mentioned above, UXP is Upbound's official enterprise-grade distribution of Crossplane. At Upbound, we have verified that EKS, AKS, and GKE can handle UXP with Official Providers (~2,200 CRDs) without issue. UXP is hardened and tested by Upbound so companies can confidently deploy control plane architectures to production.

Connecting UXP to Upbound Cloud is enabled with a free Upbound account for simplified management, and is the best thing you can do if you have run into problems running "too many" CRDs. Upbound also offers support, services and security for customers looking for help with deployment and scale.

You can gain back even more ground by making sure your Go client for Kubernetes is up to date. The Go client added some garbage collection improvements in v0.25.0, so Kubernetes clients other than kubectl should be updated to use the latest version of that library.

SERVER-SIDE ISSUES

Server-side issues are principally memory and CPU bandwidth issues.

For control plane nodes, the control plane's api-server service requires about 4MB of memory for each CRD installed into a Kubernetes cluster. This translates to 8-12GB of memory to support our typical Crossplane production deployment of 2,000-3,000 CRDs above!

More memory is more expensive - but the bigger issue is the Kubernetes clusters can become unresponsive.

The OpenAPI controller does a lot of work every time a CRD is added or updated (builds a new swagger spec, merges it with the existing specs, and then serializes that into JSON to be served by the /openapi/v2 endpoint). Adding many CRDs to the Kubernetes API server all at once can overwhelm the CPU's bandwidth.

SERVER-SIDE SOLUTIONS

Cloud vendors usually deal with this memory explosion issue through vertical scaling (adding more memory to existing nodes). The CPU issue was handled by making the OpenAPI schema computation lazy - we only do all that work again when someone asks for it by making a request to that endpoint.

If you want a more in-depth treatment of these issues, see Nic Cope's excellent blog post Scaling Kubernetes to Thousands of CRDs.

What's Next

At Upbound, we continually work with third parties (e.g. EKS, GKE, ArgoCD, Helm, etc) to ensure their clients and control planes are running the latest client and api-server releases. For example, we work closely with AKS to see if there are any other configuration changes that can be made - we want to make sure everything is fully tested and stable, and we want to continue making everything better for the community. That's why we've opened a PR in the Kubernetes project that will cut the memory used by CRDs by 35%, and improved the discovery process for Helm.

Sound interesting? Do you have any of these problems? Why not visit the Upbound Marketplace and check out UXP? Or better still, contact Upbound today - no matter where you are on your journey, we'll help you find the perfect solution to build your internal cloud platform.

35%

Decrease in memory used by CRDs

Want to learn more about Upbound?

Contact Us