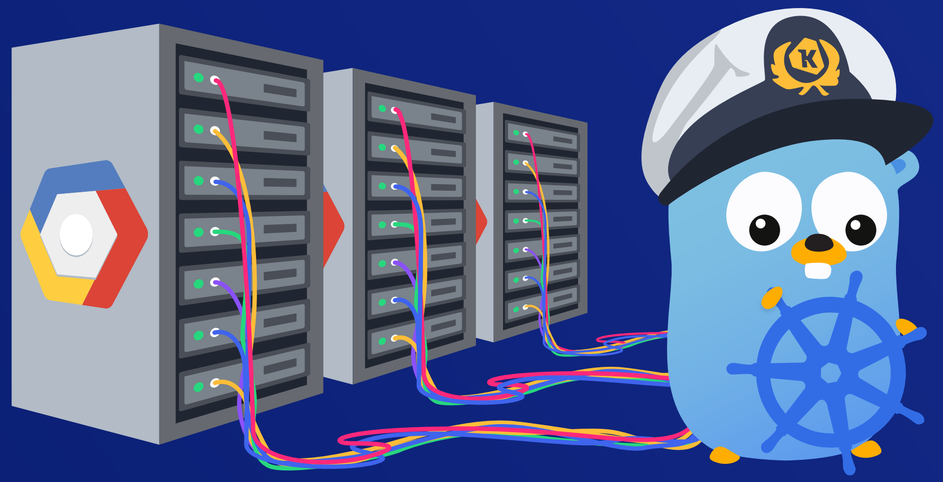


Using a Kubernetes Operator to Manage Application Tenancy in a B2B SaaS App

KOLIDE

Mike Arpaia

Co-Founder & CTO

 github.com / **marpaia**

 **marpaia** @ Kubernetes Slack

 twitter.com / **mikearpaia**

KOLIDE

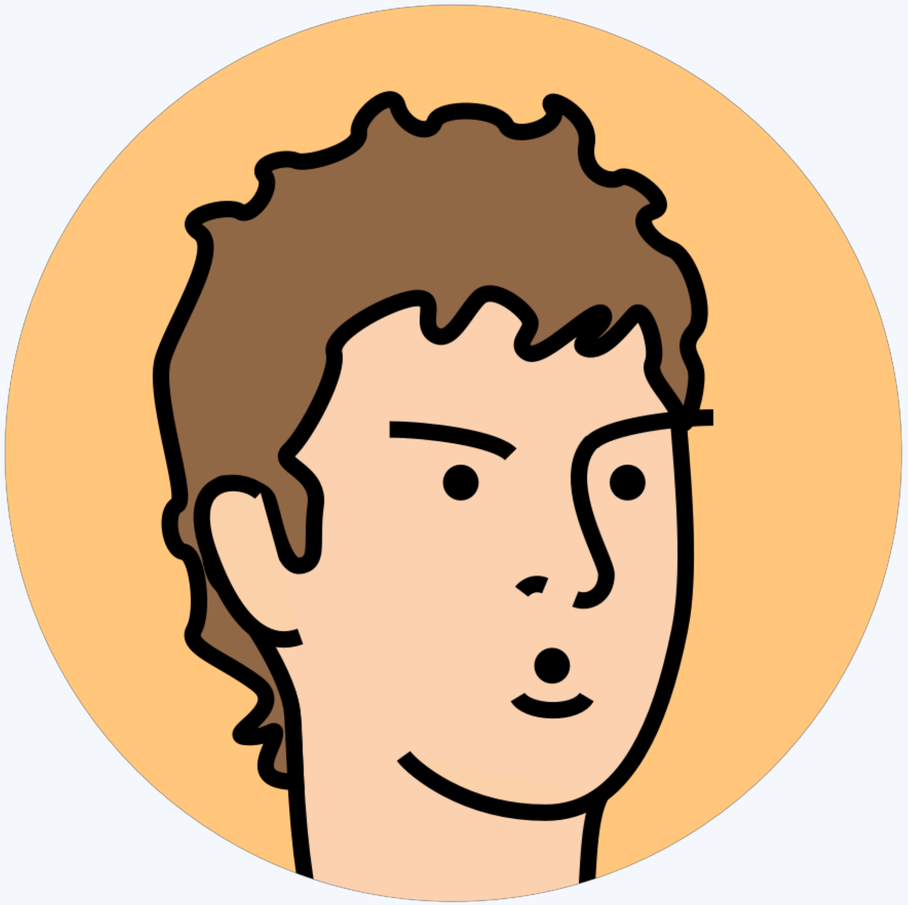
About Me

- Co-Founder & CTO of infrastructure analytics startup called Kolide
- Most recently previously worked at Facebook, Etsy, iSEC Partners
- Kubernetes Release Team from 1.11 -> 1.13
- Creator of open source tool for SQL-based security monitoring called osquery
- Enthusiastic Go Developer
- I play the bass and love the outdoors

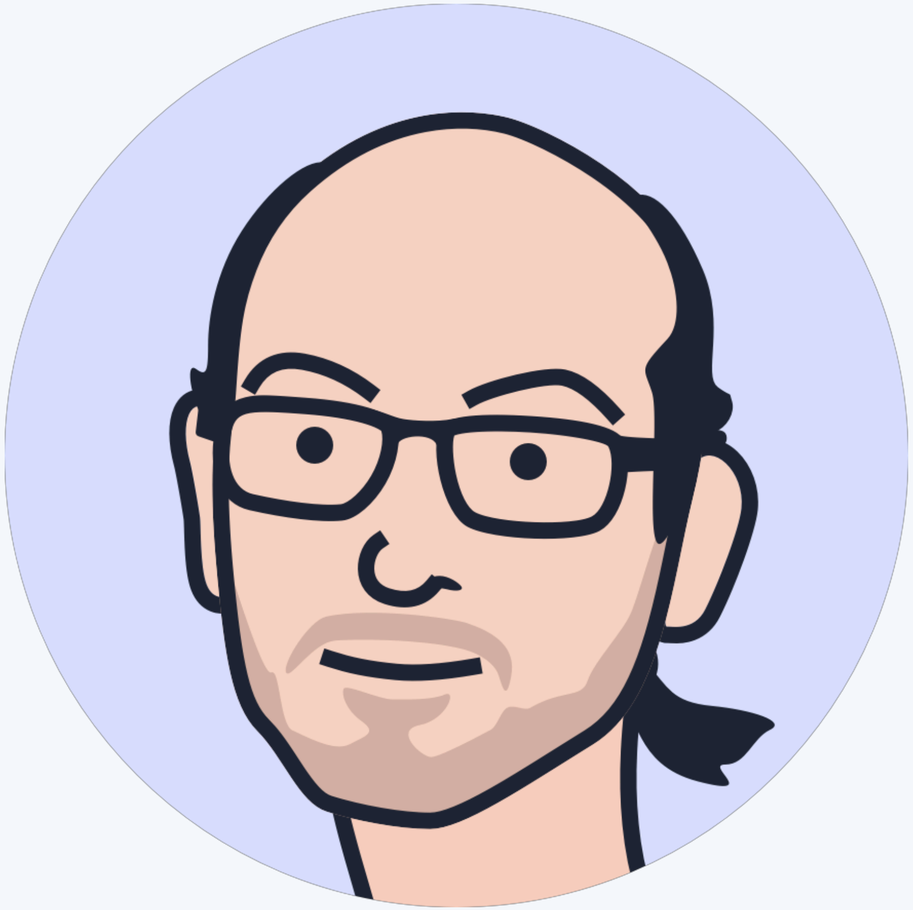
About Kolide

- Kolide is a security-first infrastructure analytics app that aims to bring total device visibility, driven by the power of osquery
- Completely SaaS B2B App where each customer trusts us with their most sensitive data
- We analyze device data to provide insights and alerts based on the health and security of your fleet

Kolide SRE



https://github.com/groob



https://github.com/directionless

Contents

- Discussion about application multi-tenancy and definition of terms
- Run down of the chosen application deployment and networking model
- Deeper dive into the Kubernetes Operator ecosystem, terms and components
- Implications of a production system: deployments, networking and security
- Navigating the tenancy space throughout the Kubernetes community
- Enumeration of lessons learned and interesting gotchas

Multi-Tenancy Concepts

KOLIDE

Application Tenancy

- Companies that create products for other companies or teams often have to reason about how to deal with the application-level tenancy of each team
- The two ends of the spectrum are to either:
  - Deploy one instance of the application which handles tenant data isolation via application logic
  - Deploy and proxy to many instances of isolated single-tenant applications
- This talk takes the second path and discusses using a Kubernetes Operator to accomplish the objective

“Hard” Multi-Tenancy

Kubernetes is the new kernel. We can refer to it as a “cluster kernel” versus the typical operating system kernel. This means a lot of great things for users trying to deploy applications. It also leads to a lot of the same challenges we have already faced with operating system kernels. One of which being privilege isolation. In Kubernetes, we refer to this as multi-tenancy, or the dream of being able to isolate tenants of a cluster.

Jessie Frazelle - https://blog.jessfraz.com/post/hard-multi-tenancy-in-kubernetes/

“Soft” Multi-Tenancy

Multiple users *within the same organization* in the same cluster. Soft multi-tenancy could have possible bad actors such as people leaving the company, etc. Users are not thought to be actively malicious since they are within the same organization, but potential for accidents or “evil leaving employees.” A large focus of soft multi-tenancy is to prevent accidents.

Jessie Frazelle - Multi-Tenancy Design Scratch Space Google Doc

Hard vs Soft Multi-Tenancy

- The journey from Soft to Hard Multi-Tenancy is a rather loose spectrum with a few key differentiating mitigations along the way
- The SaaS application multi-tenancy problem space firmly occupies the “soft” multi-tenancy classification
- Since we are the authors of all of the software we run in the cluster, our isolation needs are more around isolating data access and network traffic within tenants
- If compromise occurs, this architecture should limit unauthorized data access

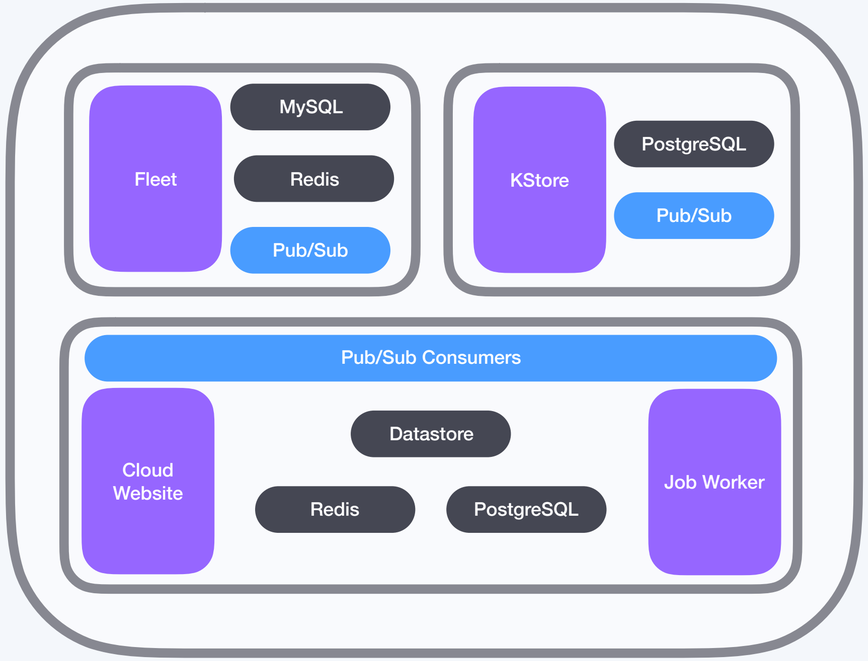
Usage Metering

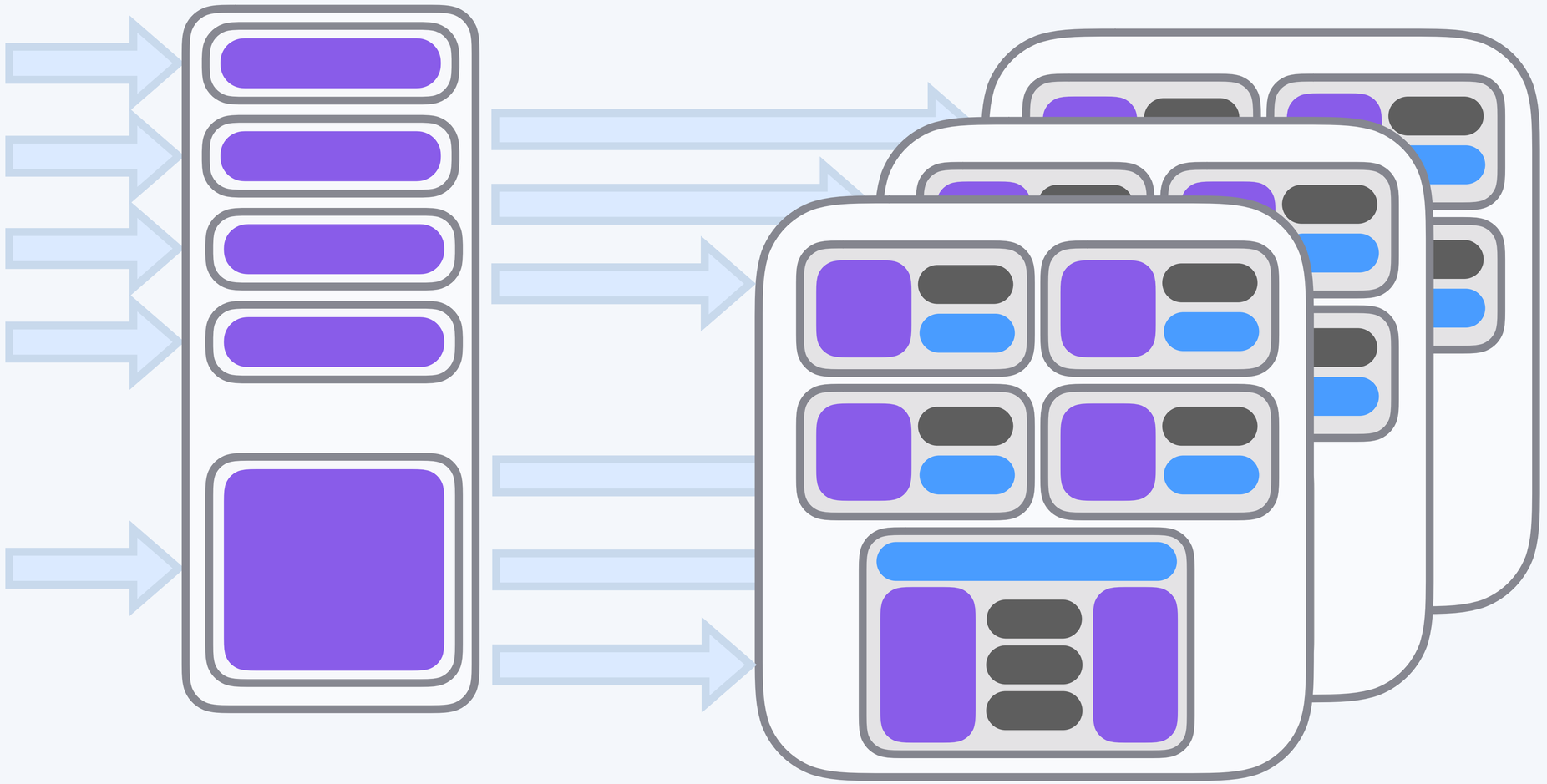
- A concept that often is discussed with multi-tenancy is “metering”
- Metering capabilities allow the operator to control how many resources that each tenant can use over a period of time
- The Kolide use-case is not concerned with metering
  - We monitor and improve performance, but we don’t punish customers for our software being resource intensive

App Architecture

Single Application Instance Per Customer

- Each customer gets an isolated instance of all application server and data dependencies
- Optimizes data and compute isolation for very sensitive use-cases
- Eliminates the need for product developers to reason about data isolation and multi-tenancy
- Re-usable infrastructure orchestration as product components get re-written in different languages
- Minimizes noisy neighbor problems





Traffic from all customers is received and quickly routed into each customer's isolated environment via a stateless, context aware edge proxy.

Managing Tenants with an Operator

- To facilitate the automated management of each tenant, we created a Kubernetes Operator capable of managing each tenant dynamically based on a set of well-defined options

- Each tenant gets its own Kubernetes namespace where all compute resources are deployed

- A higher-level control plane also exists to manage things like signups, routing, etc.

Operator Ecosystem

Kubernetes Operators

- In Kubernetes, the combination of a custom resource definition (CRD) and a controller that manages the lifecycle represented by instances of the custom resource.
- Excellent tooling with several great options for writing operators in Go
- A great place to tack event-based operations in a Kubernetes cluster

CRD + Controller

- The term ***operator*** has come to represent this combination of a custom resource definition (***CRD***) and a custom ***controller***
- For the sake of clarity I'll use the terms ***CRD*** and ***controller*** to describe the two distinct parts of the Kubernetes Operator
- The ***CRD*** represents the inputs to your system
- The ***controller*** is a server executable which interacts with the Kubernetes API server

Custom Resource Definition

- The *custom resource definition* (***CRD***) API allows you to introduce your own API into a project or a cluster and allows the Kubernetes API server to begin serving the specified custom resource.

- Source of truth is an annotated Go struct which works directly with all of the API machinery

- A neat standard to adhere to since, while there's no great way to manage all of the YAML for everyone, if you stick to the API, you'll work with a lot of the ecosystem

Custom Resource

- An instance of the CRD is called a *custom resource* (***CR***)
- Often seen as YAML
- Sometimes parameterized via tools like Helm, Ksonnet, etc.
- `kubectl get tenants`

```yaml
apiVersion: kolide.com/v1
kind: Tenant
metadata:
  name: dababe
  labels:
    name: dababe
spec:
  databases:
    postgres:
    - name: cloud
    - name: kstore
    mysql:
    - name: fleet
  email: mike@kolide.co
  organization: Kolide Inc.
  pgbouncer:
    defaultPoolSize: 10
  repos:
  - name: fleet
    container:
      name: gcr.io/kolide-private-containers/fleet
      version: cdac80a
    varz:
      ref: cdac80a
      template: tools/k8s/fleet.template
      varz: tools/k8s/varz.yaml
  - name: kstore
    container:
      name: gcr.io/kolide-private-containers/kstore
      version: 29ca464
    varz:
      ref: 29ca464
      template: tools/k8s/kstore.template
      varz: tools/k8s/varz.yaml
```

Controllers

- Controllers are long-running server processes which continuously observe current state and endeavor to converge current state and desired state by taking a variety of actions
- Commonly an eventually consistent, single-replica deployment of a single process, most often written in Go
- Extensive first-party and third-party Go library support
- The more control you need, the closer you should get to the core API machinery

Go Controller Ecosystem

- Rich, developer-friendly ecosystem simplifies writing a controller in Go
- k/sample-controller is a really great, albeit somewhat advanced, example of a controller working with the core API's
  - This is my personal favorite and what we based our controller on originally
- CoreOS created the ***Operator Framework*** to help with the creation, distribution, and execution of operators
  - A lot of features that make it easier to get started writing new operators
  - Came out after we created our operator, might use it in the future

Controller Code Examples

KOLIDE

Simplified Main

```go
// get a k8s.io/client-go/rest.Config with the provided kubeconfig flags
cfg, err := clientcmd.BuildConfigFromFlags(flMaster, kubeconfig)
if err != nil {
    return errors.Wrap(err, "error building kubeconfig")
}

// use the k8s.io/client-go/rest.Config to get a REST client which includes
// a versioned API client for Kuberneetes types
kubeClient, err := kubernetes.NewForConfig(cfg)
if err != nil {
    return errors.Wrap(err, "error building kubernetes clientset")
}

// use the k8s.io/client-go/rest.Config to get a REST client which includes
// a versioned API client for the kolide.com provided types as well
lessorClient, err := clientset.NewForConfig(cfg)
if err != nil {
    return errors.Wrap(err, "error building clientset")
}

resyncPeriod := time.Duration(flResyncPeriod) * time.Second

kubeInformerFactory := kubeinformers.NewSharedInformerFactory(kubeClient, resyncPeriod)
lessorInformerFactory := informers.NewSharedInformerFactory(lessorClient, resyncPeriod)

c := controller.NewController(
    logger,
    kubeClient,
    lessorClient,
    kubeInformerFactory,
    lessorInformerFactory,
    flBroadcastEvents,
)

stopCh := signals.SetupSignalHandler()

go kubeInformerFactory.Start(stopCh)
go lessorInformerFactory.Start(stopCh)

if err = c.Run(flWorkers, stopCh); err != nil {
    return errors.Wrap(err, "error running controller")
}
```

NewController (initialization)

```
// NewController returns a new controller
func NewController(
   // redacted for brevity

) *Controller {
   // redacted for brevity

   // Get references to shared index informers
   namespaceInformer := kubeInformerFactory.Core().V1().Namespaces()
   secretInformer := kubeInformerFactory.Core().V1().Secrets()
   deploymentInformer := kubeInformerFactory.Apps().V1beta2().Deployments()
   statefullSetInformer := kubeInformerFactory.Apps().V1beta2().StatefulSets()
   serviceInformer := kubeInformerFactory.Core().V1().Services()
   podDisruptionBudgetInformer := kubeInformerFactory.Policy().V1beta1().PodDisruptionBudgets()

   tenantInformer := tenantInformerFactory.Kolide().V1().Tenants()

   controller := &Controller{
```

NewController (event handler)

```
  controller := &Controller{
    // redacted for brevity
  }

  // Set up an event handler for when tenant resources change
  tenantInformer.Informer().AddEventHandler(
    cache.ResourceEventHandlerFuncs{
      AddFunc: controller.enqueueTenant,
      UpdateFunc: func(old, new interface{}) {
        controller.enqueueTenant(new)
      },
    },
  )

  return controller
}
```

Tenant Synchronization

```go
// resolveTenantState compares the actual state with the desired, and attempts to
// converge the two. It then updates the Status block of the tenant resource
// with the current status of the resource.
func (c *Controller) resolveTenantState(key string) error {
    ctx := context.Background()

    tenant, ok, err := c.tenantForCacheKey(key)
    if err != nil {
        return errors.Wrap(err, "ensuring tenant")
    }
    if !ok {
        level.Info(c.logger).Log("err", "attempted to process tenant but tenant no longer exists", "tenant", key)
        return nil
    }

    if err := c.validateTenant(tenant); err != nil {
        // We choose to absorb the error here as the worker would requeue the
        // resource otherwise. Since the tenant is invalid, requeueing the
        // tenant won't fix this problem.
        level.Info(c.logger).Log("msg", "tenant is invalid", "err", err, "key", key)
        return nil
    }

    generator := resources.NewGenerator(tenant)

    switch tenant.Spec.State {
    case kolidev1.TenantStateFrozen:
        return c.resolveFrozenTenantState(ctx, tenant, generator)

    case kolidev1.TenantStateArchived:
        return c.resolveArchivedTenantState(ctx, tenant, generator)

    case kolidev1.TenantStatePurged:
        return c.resolvePurgedTenantState(ctx, tenant, generator)

    default:
        // Active, Idle, New, etc
        return c.resolveActiveTenantState(ctx, tenant, generator)
    }
}
```

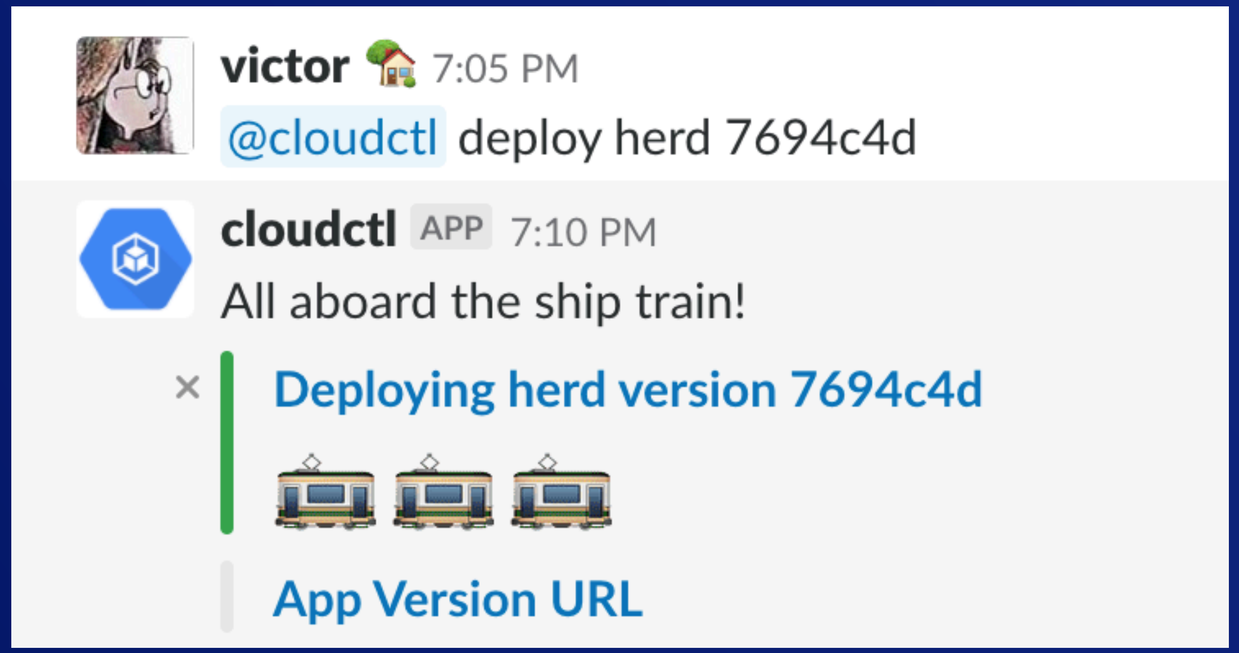
KOLIDE

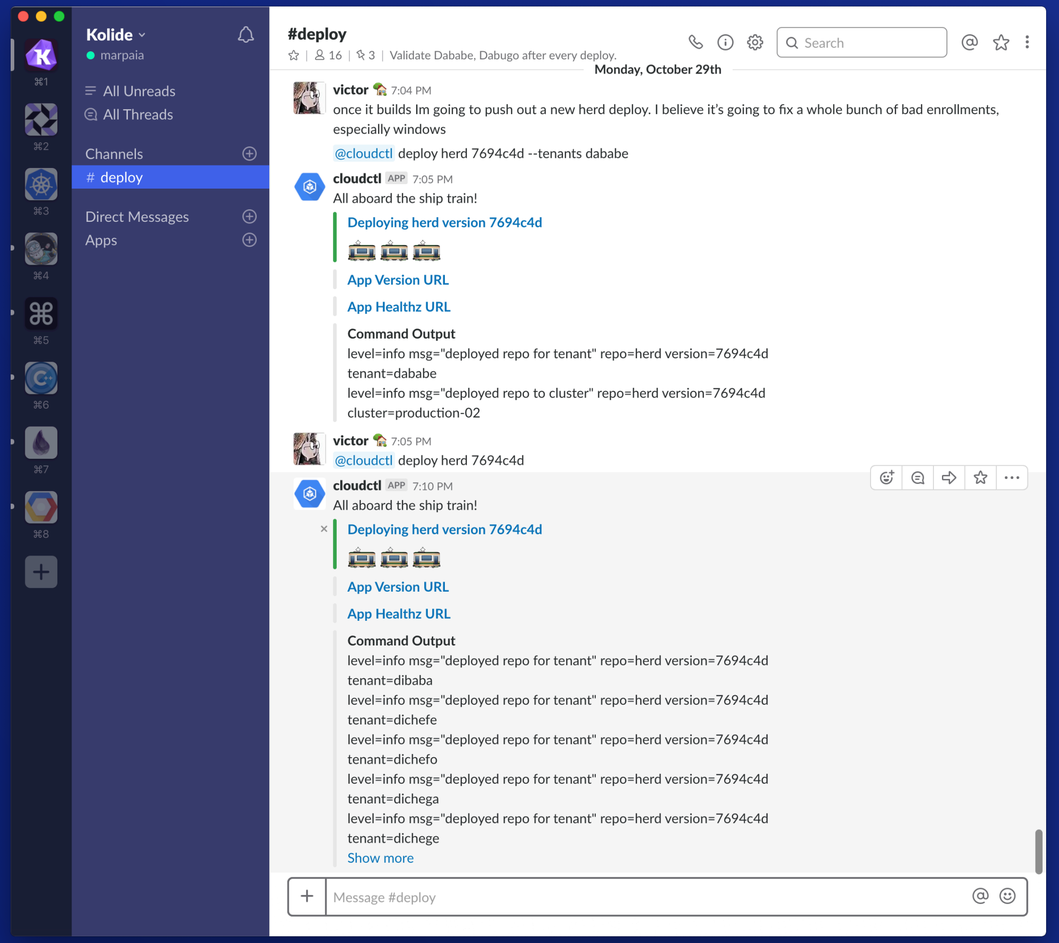
Deployments

Deployment Process

- To facilitate frequent, safe deploys, we wrote a Slack bot called ***cloudctl*** which can deploy any part of the tenant stack to any combination of tenants
- The slack bot interacts with the Kubernetes API server to update the relevant tenant custom resources
- The controller observes the changes in the tenant resources and starts making the changes
- This kind of read/change/update operation is currently prone to races without transactions or locks but this will improve with server-side apply

Slack Bot Usage





KOLIDE

Deploy Function

```go
func (c *Client) Deploy(ctx context.Context, repo, version string, tenants []string) error {
    clusterMappings := make(map[string][]string)
    // append tenants to clusterMappings if provided. Otherwise, append an empty []string to clusterMappings[cluster]
    if len(tenants) > 0 {
        for _, tenant := range tenants {
            cluster, err := c.ds.GetMapping(ctx, tenant)
            if err != nil {
                return errors.Wrapf(err, "error getting cluster mapping for %s", tenant)
            }
            clusterMappings[cluster] = append(clusterMappings[cluster], tenant)
        }
    } else {
        clusters, err := c.ds.GetAllClusters(ctx)
        if err != nil {
            return errors.Wrap(err, "error getting cluster list")
        }

        for _, cluster := range clusters {
            clusterMappings[cluster.Name] = []string{}
        }
    }

    for mapping, _ := range clusterMappings {
        cluster, err := c.ds.GetCluster(ctx, mapping)
        if err != nil {
            return errors.Wrap(err, "getting cluster")
        }

        // create a kolide client for this cluster
        kolideClient, err := c.ClientForCluster(ctx, cluster.Zone, cluster.Name)
        if err != nil {
            return errors.Wrap(err, "getting kolideClient")
        }

        // run the deployment request
        err = c.DeployWithClient(kolideClient, repo, version, clusterMappings[cluster.Name])
        switch {
        case err != nil:
            level.Info(c.logger).Log("err", err, "msg", "deploying repo to cluster")
        default:
            level.Info(c.logger).Log(“msg", "deployed repo to cluster”, "repo", repo)
        }

    }
    return nil
}
```

KOLIDE

Deploy Function Continued

```go
func (c *Client) DeployWithClient(kolideClient *clientset.Clientset, repo, version string, tenants []string) error {
    // we're going to list the tenants which need to be updated based on the
    // tenants option. to do this, we construct a label selector which will
    // identify the requested tenants.
    listOptions := metav1.ListOptions{}

    if len(tenants) > 0 {
        listOptions.LabelSelector = fmt.Sprintf("name in (%s)", strings.Join(tenants, ","))
    }

    // now we can list the tenants which are requested to be operated on
    tenantList, err := kolideClient.KolideV1().Tenants("default").List(listOptions)
    if err != nil {
        return errors.Wrap(err, "error listing tenants")
    }

    // if no tenants matched the given selector, return an error
    if len(tenantList.Items) == 0 {
        return fmt.Errorf("no tenants found")
    }

    // iterate through each tenant and update the repos based on the supplied
    // parameters
    for _, tenant := range tenantList.Items {
        for _, r := range tenant.Spec.Repos {
            if r.Name == repo {
                r.Container.Version = &version
                r.Varz.Ref = &version
            }
        }

        // update the tenant copy with the requested parameters
        if _, err := kolideClient.KolideV1().Tenants("default").Update(&tenant); err != nil {
            return errors.Wrapf(err, "error updating tenant %s", tenant.Name)
        }
    }

    return nil
}
```

KOLIDE

Networking

KOLIDE

Ingress

- For any tenant/sharding scenario, you need a central index/router for managing external ingress traffic
- There are many options available for this
  - Envoy
  - Scripting with Nginx (SRECon Talk from Shopify)
- For our needs we ended up writing a custom proxy in Go
- We like our custom proxy, although it’s a decision we might re-evaluate in the future

Internal L4 Edge Proxy

- To ingress traffic to each isolated instance of the app, we built a minimal edge proxy called “Shuffler” for North South traffic
- Similar to Envoy with regards to the separation of the control plane from the data plane
  - Control Plane is a gRPC service for managing high level configuration (with GCP Datastore for persistence)
  - Data Plane reads routing rules out of GCP Datastore
- Adds 1-2 ms of latency to each request

Intra-Namespace Traffic

- Per-tenant, service to service traffic uses the namespace-local service name as the DNS address
- This allows each tenant to talk to it’s own instance of each service via consistent DNS which minimizes configuration change between tenants
- Network Policies can be used to enforce the desired level of isolation

Security

KOLIDE

Secret Distribution

- Each tenant needs two kinds of secrets
  - Secrets that are the same across all tenants (API keys, etc)
  - Secrets that are unique per tenant (database credentials, etc)
- Kubernetes secrets are not able to be shared across namespaces
- Since each tenant is in it’s own namespace, a complete set of secrets must be copied to each namespace on an as-needed basis

Secret Distribution

- Tooling pulls secrets from external storage and populates a reserved template namespace in each cluster
- The controller uses the shared informer libraries to maintain an in-memory cache of every secret in every namespace
- When a tenant is synchronized, secrets are copied from the template namespace to the tenant's namespace
  - This process always checks whether or not the secrets needs to be updated via the in-memory cache before communicating with the Kubernetes API server directly
- Finally, any tenant specific secrets (DB credentials) are copied directly from storage if necessary

Secret Distribution - External Storage

```go
package secret

import (
  "context"

  corev1 "k8s.io/api/core/v1"
)

// Store is the interface which defines the controllers interactions with an
// arbitrary exo-cluster secret storage mechanism.
type Store interface {
  Get(ctx context.Context, namespace string, name string) (*corev1.Secret, error)
  List(ctx context.Context, namespace string) ([]*corev1.Secret, error)
  Put(ctx context.Context, s *corev1.Secret) error
  Delete(ctx context.Context, namespace, name string) error
}
```

Secret Distribution - External Storage

- At Kolide, we are extremely heavy users of Google Cloud Platform (GCP)
- Our implementation of the secret.Store uses two GCP products
- Google Key Management Service (KMS) is used for access-controlled encryption and decryption of secrets by services (like the controller)
- Google Cloud Storage (GCS) is used to persist and retrieve encrypted secrets

Secret Distribution - Synchronization Loop Performance

- When a controller first starts, it must run a synchronization on all tenants to ensure that current state is desired state
- When nothing has changed, a complete tenant synchronization should complete extremely quickly (a few milliseconds)
- This precludes making any calls to the API server during the “happy path”
- Optimizing the performance of the run loop was very much an exercise in efficient secret distribution
- Heavy use of in-memory caching via the shared informer API made this possible

Secret Distribution - K8s Secrets Security

- While the security story around encryption at rest with K8s secrets leaves much to be desired, we have accepted this risk for a few reasons
- By committing to the standard, we will inherit security improvements that are in development for GKE and Kubernetes in general
- The attack surface of the API Server's etcd is limited in GKE
  - See the talk on GKE internal security by Aaron, Greg, and CJ from NEXT

Stuff We've Learned

Performance and Scalability

- Initial optimizations involved reducing the number of API requests to the API server in the controller
  - This went from being a huge problem to a non-issue over time
  - The work on secret distribution with shared informers was a huge win
- When we reached 750-1000 tenants, several different aspects of our cluster started to fall over
  - It became apparent that this was our “one cluster maximum”
  - We solved this problem with a multi-cluster deployment architecture

Optimizing for Customer Time-To-Value

- Since each tenant had independent data requirements, it could take several minutes to create the database instances and run all of the migrations for a new tenant
- Customers need to be able to sign-up and immediately start getting value out of the app
- To solve this, we created a mechanism in the controller which would maintain a configurable number of “spares” (unallocated tenants)
- Sign-up then synchronously “allocates” a spare and drops the user’s web session into it

Incorporate health checking into the controller

- During tenant synchronization, we must interact with the API Server to deploy resources, adjust replica counts based on load, etc
- When we first started, we would make the API requests and move on quickly
- This caused the API Server to fall over due to the rate of API requests while it was busy communicating with the Kubelets
- We solved this by not completing a synchronization until all desired actions were observed to be completed
- Thus, by adjusting the work queue size, we had fine control over the number of tenants that would ever be operated on at once

Situations when this architecture works really well

- You have a small number (hundreds to thousands) of large/active customers
  - You must be able to take advantage of the fact that tenants can be scaled independently but must be at least large enough for the minimal tenant footprint to be profitable
- You have application components written in several different languages
  - Assembling tenants via the CRD is productive in polyglot environments
- You can take advantage of the independent scaling of each tenant
  - Small customers must be large enough to make your smallest footprint profitable
  - Large customers must be able to be handled by scaling out tenant service replicas

Situations when this architecture doesn’t work well

- You have a large number of small/inactive customers
  - If this is the case for you, the economics of a traditional multi-tenant application will work better for you
- Your data security requirements aren’t very stringent
  - These isolations principles probably aren’t worth the effort or expense if your application data isn’t extremely sensitive
- You don’t like writing Go
  - Writing an operator to encapsulate your infrastructure objective turns the problem into a Go software development problem

What are some general benefits of this approach?

- Extremely reliable tenant data isolation
- Outages in tenant components are usually isolated to a single customer
- Easy to test out new versions of different applications with different customers
- Decouples infra teams from product development teams
- The CRD is a clean, typed interface for describing customer configuration
- The technology is fun and exciting

What are some pain points of this approach?

- You must maintain a strict culture of articulating operations tasks in controller code
- Most people don’t bend Kubernetes in this direction
- Running 90+ pods per node starts to make things choke, despite the current max being 110
- CRDs don’t do well at representing multi-cluster concepts
- Accurately monitoring every component of every tenant is challenging
- No server-side apply results in error-prone home-grown alternatives

Community Efforts

Multi-Tenancy Working Group

- Working Group led by David Oppenheimer (Google) and Tasha Drew (VMWare), previously also led by Jessie Frazelle (Microsoft)
- A lot of work on definition of terms and establishing group consensus
- A lot of great documents written by David and Jessie
- See the draft of the charter in this Google Doc
- Slack: #wg-multitenancy
- Google Group: kubernetes-wg-multitenancy@googlegroups.com

Lessor - https://github.com/lessor/lessor

- My shell of a controller for working on open-source multi-tenancy concepts
- Adds a “Tenant” CRD in the “lessor.io” API group
- Not under very active development, but a good place to start collaborating if anyone is interested in working on this stuff with me!

Questions?

github.com / **marpaia**

marpaia @ Kubernetes Slack

twitter.com / **mikearpaia**