Building Hybrid cloud with Kubernetes

Alena Prokharchyk, Principal Software Engineer
@RancherLabs
Data Center infrastructure evolution timeline

The data center of the future
Containers started shaping infrastructure in 2013

Containers are

- Lightweight and portable
- An easy way to pack, ship, and run software
- Provide secure isolation for user applications
You start with deploying an application on a single host.

- can be a developer laptop
- or a VM
But then realize your application is more complex
and it has to be scheduled on multiple hosts
You need an orchestrator

- To manage application scheduling
- To configure and monitor application health
- To make application upgrades simple
proved to be the most popular orchestrator
But nobody said Kubernetes is easy
Orchestrating orchestrator was challenging

- Installation was hard
- Upgrade was even harder
- Setup requirements varied depending on a cloud

Kubernetes The Hard Way

- A Github repository created by Google engineer Kelsey Hightower (@kelseyhightower)
- Manually configure the cluster step-by-step
- The tutorial is based on Google Cloud Engine
- The best way to learn Kubernetes
  - Showing the distributed nature of a Kubernetes cluster
  - HA for master nodes
  - Authentication method by bootstrap tokens and certificates/keys
But it got better

A lot of Kubernetes installation tools emerged to address the challenge

Kubernetes as a public offering was made available on several clouds

kops kubespray kubeadm
Is the puzzle complete?

Cloud provider orchestrates Kubernetes cluster

Kubernetes orchestrates an app to a single data center

Docker deploys application on a host
Nowadays you need more than one k8s cluster
Use case #1 - Geographical separation

- Clusters resides in different regions
- Front faced by Load Balancing with GEO Routing support
- Note: K8s components like etcd/kubelet/apiserver can’t be too far apart as the connection between has to be reliable. That drives the need for this kind of separation
Use case #2 - Logical separation driven by security reasons

- Given k8s is not hard multi-tenant, if the API server is compromised, the whole cluster is compromised.

- Having a cluster per team or per project help to reduce the problem occurrence.

- Different level of security checks and monitoring can apply to different clusters.
Use case #3 - Logical separation driven by functionality reasons

• Organization that has different groups explore k8s on their own.

• Different teams have their own best practices.

• Different practices means different cluster configuration
3 challenges to address

1. **Centralized** K8s installer/operator/manager that works everywhere

2. **Centralized** authentication/authorization access to multiple clusters

3. **Centralized** add-on manager for k8s system services like DNS/Network/Ingress
...and build an **open source** platform to manage multiple Kubernetes clusters.
Challenge #1 - Provisioning
Provisioning use cases

• **Requirement:** Create Kubernetes cluster supporting Kubernetes as a public offering (GKE, EKS). **Implementation:** Integrate with k8s provider APIs to create/remove/update cluster

• **Requirement:** Rancher provisions the hosts in the cloud, and installs Kubernetes on them. **Implementation:** a) Develop, or integrate with existing machine driver to provision the VMs b) Chose an existing k8s third party installer, or develop your own

• **Requirement:** User provisions the hosts, and Rancher installs Kubernetes on them. **Implementation:** Chose an existing k8s third party installer, or develop your own
Chosing Kubernetes installer

To provision the VMs, integration with Docker machine drivers was goto solution

Deciding on k8s installer was not as easy:

- Installers like kops were no-go as they work on certain providers.

- Generic installers like kubeadm, kubespray were evaluated, and each has its own set of problems.
Decision was made

To write our own **open source** Kubernetes installer that:

- Can be used to deploy k8s anywhere
- Can run as a standalone tool, and as a driver
- Launches Kubernetes in docker containers
- Has the most simple configuration
- Written in Go
After that, write a layer integrating with all Kubernetes drivers

A tool like docker-machine, to provision Kubernetes clusters for different cloud providers
Build a management layer on top

API/Management layer would:

• Provision the clusters by calling kontainer-engine driver

• Monitor provisioned clusters’ health

• Configure cluster addons by talking to k8s cluster directly
Rancher 2.0, version 1

- The initial Rancher 2.0 server was written in **Java**, having **mysql** as a database

- Rancher agent was deployed as a Kubernetes application on publicly hosted k8s clusters, and as a docker container on regular hosts

- Agent was reporting back the state of K8s cluster, as well as certain Kubernetes events
The design was flawed

• A lot of Java to Go conversion as Docker, Kubernetes, RKE, Kontainer-engine are written in Go

• A big diversion between management layer code and code talking to user clusters

• A big split between Rancher API and Kubernetes APIs implied back and force conversion
Rancher 2.0, version 2

- Built on top of Kubernetes, with **etcd** as a database

- Written in **Go**

- Rancher API extending Kubernetes API, allowing to use existing concepts while introducing our own

- Every Rancher resource is Kubernetes **CRD** (Custom resource definition)

- Functional components running as Kubernetes **controllers**
What is CRD?

A way to extend Kubernetes API server
What is a custom controller?

In a few words, controller:

- Watches for the resource changes
- Executes some custom logic based on the resource spec or/and status
- Updates the resource status with the result
The big picture

- Secrets propagation
- Public endpoints population
- NetworkPolicy configuration

RANCHER SERVER

- Management controllers
- User cluster 1 scoped controllers
- User cluster 2 scoped controllers
- User cluster 3 scoped controllers

- Cluster provisioner
- Cluster stats
- Cluster healthcheck
- Authentication
- Catalog manager
- ...

etcd

User cluster 1 API Server

User cluster 2 API Server

User cluster 3 API Server
Lessons learned

• Be careful when chose to use a third party software for the key layer of your product over writing your own

• If possible, release the part of the project as a standalone tool, so it gets wider adoption and testing

• When writing an integration software, consider using the same development stack as the project you are integrating with
Challenge #2 - Authentication and Authorization cross clusters

WE HEARD YOU LIKE AUTHENTICATION SO WE PUT AUTHENTICATION IN YOUR AUTHENTICATION

SO YOU CAN AUTHENTICATE WHILE YOU AUTHENTICATE
Things to address

- To have a unified authentication experience across different types of clusters
- Manage users and permissions across clusters
- Prevent users from gaining too much access to the underlying infrastructure resources in environments of low trust
Unified Authentication

- Rancher’s authentication proxy sits outside and independent of the clusters

- Administrators need only configure their authentication provider once in Rancher and it will be applied everywhere

- Rancher validates and forwards the request to the appropriate cluster using user impersonation
RBAC Authorization: Kubernetes native approach

Introducing RBAC:
• Per-namespace/ resource, role, action

Examples:
• *Alice* can list *Eng* services, but not *HR*
• *Bob* can create Pods in *Test* namespace, but not in *Prod*
• *Scheduler* can read *Pods* but not *Secrets*
RBAC Authorization in Rancher

- Extends Kubernetes RBAC
- Introduces a concept of a **Project** to allow namespaces grouping
- Self service access - once user is added to the project, it automatically inherits all the permissions
Controller model proved to be the best one when it came to writing Authentication/Authorization piece
Example of Rancher RBAC controller

• The controller listens on namespace creation in user cluster

• Derives permissions from Rancher Project the namespace belongs to, pushes them down to user cluster level and sets them on the namespace
Once authorization is configured, it’s time to protect infrastructure level access.
Base use cases for Pod Security Policy

- Prevent Pods From Running With Root Privileges
- Prevent Pods From Accessing Certain Volume Types
- Prevent Pods From Accessing Host Ports

<table>
<thead>
<tr>
<th>Control Aspect</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running of privileged containers</td>
<td>privileged</td>
</tr>
<tr>
<td>Default set of capabilities that will be added to a container</td>
<td>defaultAddCapabilities</td>
</tr>
<tr>
<td>Capabilities that will be dropped from a container</td>
<td>requiredDropCapabilities</td>
</tr>
<tr>
<td>Capabilities a container can request to be added</td>
<td>allowedCapabilities</td>
</tr>
<tr>
<td>Controlling the usage of volume types</td>
<td>volumes</td>
</tr>
<tr>
<td>The use of host networking</td>
<td>hostNetwork</td>
</tr>
<tr>
<td>The use of host ports</td>
<td>hostPorts</td>
</tr>
<tr>
<td>The use of host's PID namespace</td>
<td>hostPID</td>
</tr>
</tbody>
</table>

With Rancher, we make user experience easier by

- Configuring Pod Security Policy templates that can be used across clusters to restrict the types of pods that can be deployed in a cluster.

- Pod Security Policy can be applied to the entire cluster as well as an individual project.
Network isolation using Kubernetes Network policy

• By default, all pods in Kubernetes can communicate with each other

• In real world, you’d like to limit the communication based on certain rules

• Network policy is an easy way to configure the rules

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
podSelector:
  matchLabels:
    app: web
ingress: []
```
Rancher leverages network policies to support project based multitenancy
Lessons learned

• Try extending, not replacing, core functionality of the system you are integrating with

• Consider all the ways to make admin work easier - templatizing frequently repeated configs is the example
Challenge #3 - Managing Kubernetes addons

I INTEND TO RUN CRYISIS ON KUBERNETES

MAKE IT SO
Kubernetes addons

- Addons extend the functionality of Kubernetes.
- Get deployed as a native Kubernetes apps
- Upgrading addons doesn’t require upgrading Kubernetes (in most cases)
Most popular addons

- Network CNI drivers (Calico, Flannel, etc)
- DNS (CoreDNS, SkyDNS)
- Ingress controller (Nginx, Traefik, Haproxy)
- Dashboard (UI)
Addons on Public Kubernetes providers are

• Pre-set
• Not advisable to modify
• Upgrade is not transparent

It may look like I'm flexible but really I'm just stuck and in pain..... lots of pain.
We wanted to install critical addons by default, but also give an advance user enough flexibility to change them
Addons on Kubernetes clusters provisioned by Rancher, are

- Configurable
- Can be changed after the fact, if user changes the mind about the initial choice
- Can be disabled, if user wants to install his own custom addons
What if user wants more fine grained control over addons

• They can be installed from Global catalog from a Helm Chart
• Upgrades pushed for the charts, will become available to any setup using the chart
• Chose this option if you want more control over your addons
And we followed it in every aspect of the app.

- No cloud provider lock in: cluster can be provisioned anywhere in the cloud or on prem
- No Rancher API/UI lock in: any cluster provisioned by Rancher, can be accessed directly via kubectl
- Rancher extends - not overrides - Kubernetes native functionality
• Be careful when chose to use a third party software for the key layer of your product over writing your own
• If possible, release the part of the project as a standalone tool, so it gets wider adoption and testing
• When writing an integration software, consider using the same development stack as the project you are integrating with
• Setup some nice defaults, but give enough flexibility to suite more complex use cases
• Try extending, not replacing, core functionality of the system you are integrating with
KEEP CALM ITS DEMO TIME