RANCHER

September 2017

Rancher 2.0:
Technical Architecture

RANCHER

Contents

Background

We developed the Rancher container management platform to address the need to manage containers in production. Container technologies are developing quickly, and as a result, the Rancher architecture continues to evolve.

When Rancher 1.0 shipped in early 2016, it included an easy-to-use container orchestration framework called Cattle. It also supported a variety of industry-standard container orchestrators, including Swarm, Mesos, and Kubernetes. Early Rancher users loved the idea of adopting a management platform that gave them the choice of container orchestration frameworks.

In the last year, however, the growth of Kubernetes has far outpaced other orchestrators. Rancher users are increasingly demanding better user experience and more functionality on top of Kubernetes. We have, therefore, decided to rearchitect Rancher 2.0 to take advantage of the popularity of Kubernetes by rebasing the popular Rancher experience (formerly known as Cattle) on Kubernetes.

Kubernetes is everywhere

When we started to build Kubernetes support into Rancher in 2015, the biggest challenge we faced was how to install and setup Kubernetes clusters. Off-the-shelf Kubernetes scripts and tools were difficult to use and were unreliable. Rancher made it easy to setup a Kubernetes cluster with a click of a button. Better yet, Rancher enabled you to setup Kubernetes clusters on any infrastructure, including public cloud, vSphere clusters, and bare metal servers. As a result, Rancher quickly became one of the most popular ways to launch Kubernetes clusters.

In early 2016, numerous off-the-shelf and third-party installers for Kubernetes became available. The challenge was no longer how to install and configure Kubernetes, but how to operate and upgrade Kubernetes clusters on an on-going basis. Rancher made it easy to operate and upgrade Kubernetes clusters and its associated etcd database.

By the end of 2016, we started to notice that the value of Kubernetes operations software was rapidly diminishing. Two factors contributed to this trend. First, open source tools such as Kubernetes Operations (kops) have reached a level of maturity that made it easy for many organizations to operate Kubernetes on AWS. Second, Kubernetes-as-a-service started to gain popularity. A Google Cloud user, for example, no longer needed to setup and operate their own clusters. They could use Google Container Engine (GKE) instead.

The popularity of Kubernetes continues to grow in 2017. The momentum is not slowing. We believe that, in the not so distant future, Kubernetes-as-a-service will be available from all infrastructure providers. When that happens, Kubernetes will become the universal infrastructure standard. DevOps team will no longer need to operate Kubernetes clusters themselves. The only remaining challenge will be how to manage and utilize Kubernetes clusters, which are available everywhere.

Rancher 2.0: built on Kubernetes

Rancher 2.0 is a complete container management platform built on Kubernetes. Figure 1 illustrates the capabilities of Rancher, including:

1. Rancher Kubernetes distribution which powers the embedded Kubernetes master and enables easy setup and operations of Kubernetes clusters.
2. Ability to import existing Kubernetes clusters built using other tools like kops or operated by cloud providers like GKE.
3. Management for Kubernetes clusters across any infrastructure, including public cloud, private cloud, virtualization clusters, and bare metal servers.
4. Multi-cluster management across both embedded clusters and imported clusters.
5. Intuitive user experience built on Kubernetes.

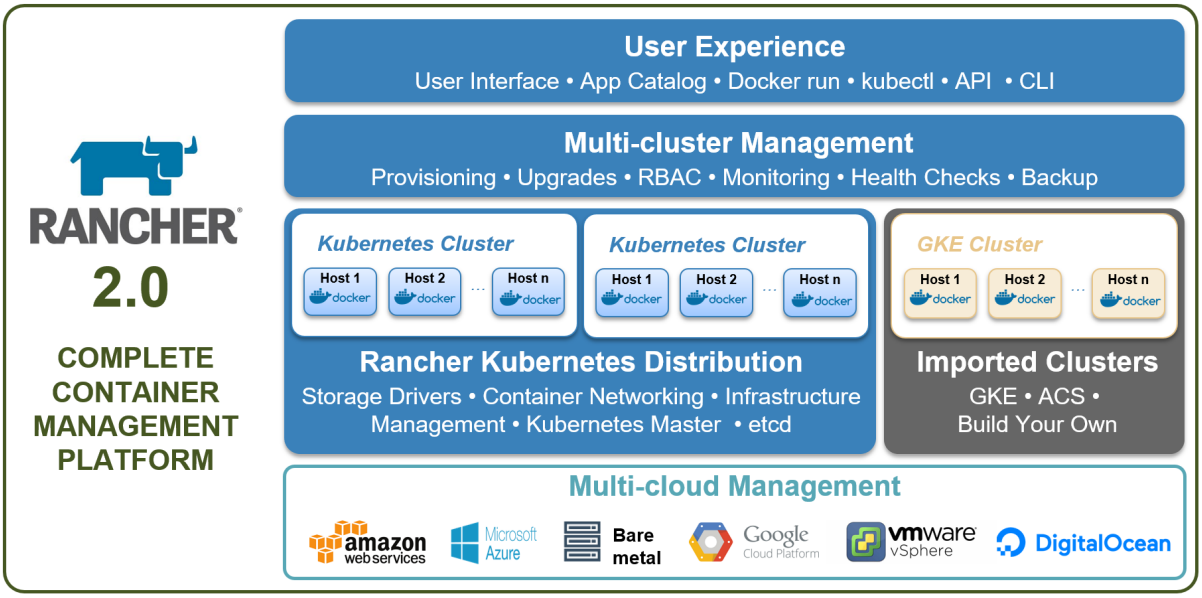

Figure 1 Overview of Rancher 2.0

High-level architecture

Like Rancher 1.0, Rancher 2.0 software consists of a server and an agent. Rancher server includes all the software components used to manage the entire Rancher deployment. Rancher agent runs on all the hosts under management in both embedded and imported clusters.

Unlike Rancher 1.0, however, Rancher 2.0 includes an embedded Kubernetes master. For example, as soon as you start Rancher using the command

```
docker run -d -p 8080:8080 rancher/server
```

you immediately have a Kubernetes cluster up and running. You don't need to perform further steps to create the first Kubernetes cluster.

Figure 2 illustrates the high-level architecture of Rancher 2.0.

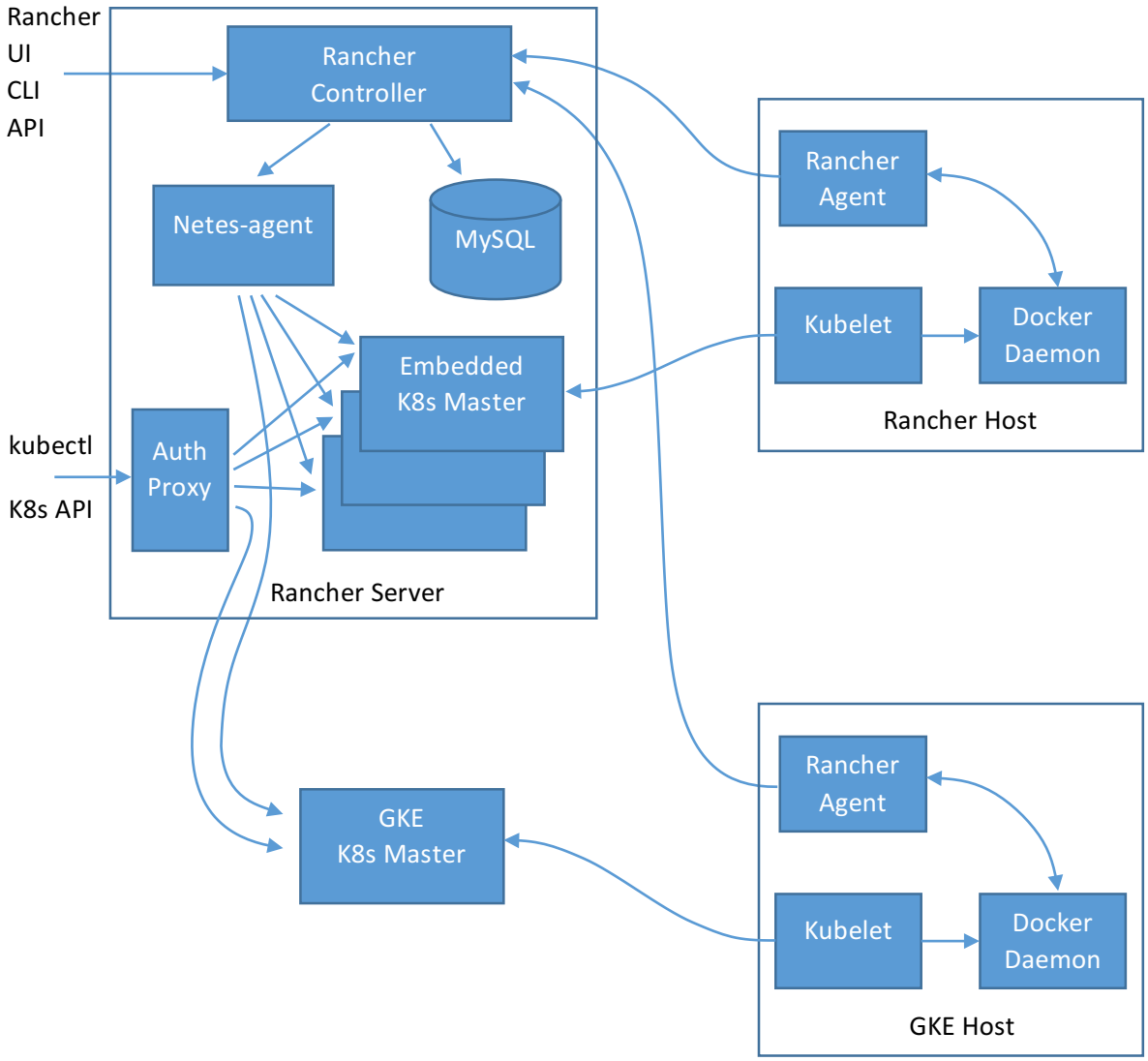

Figure 2 Rancher 2.0 High Level Architecture

Rancher controller implements the Rancher API. It includes a lightweight orchestration engine that implements Rancher constructs:

Rancher Construct	**Implementation**
Cluster	Kubernetes cluster
Environment	Kubernetes namespace
Stack	A collection of services that form an application
App	A stack created from a Rancher catalog item
Service	A group of containers with health check, DNS name, start order, upgrade policy, and scale factor. This is one Rancher construct that encompasses many Kubernetes concepts such as service, replication controller, stateful set, and deployment.
Container	Kubernetes pod

Sidekick	A multi-container pod

The Rancher controller instantiates embedded Kubernetes clusters or imports existing Kubernetes clusters like GKE.

The Netes-agent takes definitions of Rancher containers and implements them as Kubernetes pods. The Netes-agent connects to all Kubernetes clusters under management.

For example, when the user deploys a Rancher service consisting of three containers, the user makes an API call into the Rancher controller. The Rancher controller interprets the service definition and issues a request to Netes-agent to create three pods. The Netes-agent forwards the pod creation request to the appropriate Kubernetes master. The Kubernetes master writes the definition of the three pods into the Kubernetes database. The Kubernetes scheduler then associate the appropriate hosts with these pods. Finally, Kubelets launch pods on these hosts.

But what happens when there is an unexpected state change of one of the pods? For example, what if one of the pods crashes? That information will be propagated back simultaneously along two paths:

1. The Kubelet will detect the pod crash and issue a status update to the Kubernetes master.
2. The Rancher agent listens to Docker daemon events and issues a status update to the Rancher controller.

The Netes-agent is not responsible for propagating the pod status between the Kubernetes master and Rancher controller.

Embedded Kubernetes Master

The embedded Kubernetes master is a customized Kubernetes distribution consisting of the following components:

1. API Server
2. Scheduler
3. Controller Manager

To minimize memory footprint, these components are all packages in a single process within a single container. Two additional optimizations are also implemented in the embedded Kubernetes master:

1. Multi-tenant API server: for every new embedded Kubernetes master, Rancher creates a logical instance within the same process.
2. Shared DB backend for the API server: the Multi-tenant API server will by default place the database backend for multiple clusters into the same database. This greatly simplifies how Rancher manages embedded Kubernetes clusters.

Note that users can always setup standard etcd-backed single-tenant API servers by running a standard Kubernetes configuration tool like kops and then import the cluster into Rancher.

Imported Kubernetes Clusters

When a user wants to import an existing Kubernetes cluster into Rancher, they run kubectl on the imported cluster using a yaml file hosted on the Rancher server.

The user needs to have cluster admin access, so he can create a new service account which also has cluster admin access. The import process goes through the following steps:

1. The import process creates a service account in the Kubernetes cluster.
2. The import process runs a one-time pod that is configured to use the service account, reads the token associated newly created service account, and reports that token back to Rancher server. Rancher server can use that token in all future communications with that Kubernetes cluster.
3. The import process deploys the Rancher agent as a daemon set.
4. The one-time pod registers the Kubernetes cluster with Rancher server.
5. Rancher server does not save the API endpoint of Kubernetes cluster. Rancher agent talks to the Kubernetes API service through the built-in naming service.
6. When Rancher agent starts up, it registers the hosts in Rancher server.
7. Rancher agent back-populates containers launched outside of Rancher into Rancher DB using, for example, kubectl.
8. Netes-agent will receive an event that a new Kubernetes cluster is imported.
9. Netes-agent will watch all pods in all namespaces and keep that information in memory.

Rancher Controller

The Rancher controller consists of the following components:

1. Websocket proxy: forwards requests from Rancher agent or Rancher UI to the core controller.
2. Compose executor: interprets and executes the compose files.
3. Core controller: implements Rancher constructs like services, load balancers, and DNS entries into Kubernetes pods.

Core controller is the only component in Rancher written in Java. All other components are written in Go.

Rancher Agent

Rancher agent connects to Rancher server through the Websocket proxy. Websocket proxy listens on port 8080, and connects to Rancher server at port 8081.

The Websocket connection enables the agent to perform full duplex communication with the server. In addition, console sessions, stats, and log entries can also be streamed from the host to Rancher server.

The agent sends a status packet to the serve every minute. The status packet contains a listing of all the containers and their status. Multiple missed status packets cause the host to enter “reconnecting” state.

Rancher Metadata

A unique feature of Rancher is it provides EC2-style metadata to each container. By issuing an HTTP request to the link-local address 169.254.169.250, each container can obtain read-only access to the current environment and discover information such as other services, containers, hosts, and their IP addresses. The Rancher metadata service provides an easy-to-use introspection API for containers.

Rancher metadata is implemented by two components:

1. Metadata agent runs as a Rancher global services on every host
2. Rancher controller maintains an in-memory model of metadata

Rancher metadata is propagated from the Rancher controller to all the metadata agents. Rancher implements delta sync using a generation ID. Rancher controller periodically wipes out the generation ID and performs a full sync. Metadata client can request full sync. Metadata client also retains a local copy of data on disk. A restart of the metadata client therefore does not automatically trigger a full sync, as long as generation IDs still match.

Rancher controller constructs metadata for containers created through Rancher API. For containers created directly through Kubernetes API or CLI, Rancher back-populates metadata as follows:

Docker daemon → Rancher agent → Rancher server → Rancher DB → metadata (with IP, mac, etc.)

Back population works the same way for both imported and embedded Kubernetes clusters.

Rancher DNS and Load Balancer

Rancher DNS server is deployed on every host at link local address 169.254.169.250. Rancher DNS server implements a simple form of service discovery within the environment. It implements an additional level of separation based on Rancher concepts such as stacks and links.

We plan to replace Rancher DNS implementation with CoreDNS in the future.

Currently Rancher load balancer is based on HAproxy. We plan to refactor the code such that:

1. Rancher load balancer will be used as a cluster load balancer for cluster-local traffic.
2. Kubernetes ingress controller will be used for Internet-facing traffic.

Storage

Rancher volumes maps to a Kubernetes PersistentVolumeClaim (PVC). Rancher manages three types of volumes that differ in lifecycle:

1. Container scoped: the volume is created when the container is created, and destroyed when the container is destroyed.
2. Service scoped: the volume is created when the service is created, and destroyed when the service is destroyed.
3. Environment scoped: the volume is retained as long as the environment is retained.

Volumes have three access modes:

1. MultiHostRW: a volume can be accessed from multiple hosts.

2. SingleHostRW: a volume can be accessed from a single host.
3. SingleInstanceRW: a volume can be accessed from a single container.

Volumes are allocated from two types of storage pools: local and cluster-wide.

Rancher supports volumes-from. Two containers with a volumes-from relationship must reside on the same host.

Health check

Rancher health check is implemented using two mechanisms working together:

1. The built-in Kubernetes pod health checker.
2. A distributed network-based health checker.

To implement, deploy an HAproxy instance on every host, then randomly choose up to three HAproxy instances to issue health check requests. Unless there is only one host, avoid using the HAproxy instance on the same host. Service is healthy as long as at least one HAproxy instance reports success.

Networking

When Rancher originally started out there were no networking plugin interfaces for containers. There also were not many networking solutions. Rancher invented its own overlay networking solution using IPSec. This managed IPSec overlay networking solution was very attractive to most users as it allowed them to bring in nodes/hosts from various cloud providers and different regions without worrying about network security. Since the IPSec solution is built on the top of Strongswan, we can provide the most advanced—as well as the latest and greatest—encryption possible.

As users started deploying Rancher in private data centers, there was a demand for faster overlay networking solution without encryption. We therefore implemented VXLAN-based overly solution.

Before CNI was adopted as a CNCF project, Rancher adopted it and provided it as an option to users to bring in any third party CNI networking solutions.

With 2.0, users continue to have the option of using either the Rancher IPSec or VXLAN overlay solutions or use any of the third party CNI networking plugins in embedded clusters. Since an imported cluster already has a networking plugin plugged in, Rancher will support that by default and will not force the user to install Rancher networking solutions. Users can continue to use their choice of networking plugin.

A few microservices running in Rancher are responsible for making networking seamless to users. The following provides a summary of what each microservice is responsible for:

1. **`network-manager`**: This microservice enables the containers to use Rancher's Service Discovery and DNS by programming `/etc/resolv.conf`.
    a. By default containers are in their isolated networking plane and cannot access the internet. Network-manager sets up various iptables rules to enable DNAT for the containers to leverage the host network, allowing outbound internet connections for the containers.

b. Since containers are running on the overlay network, the services provided by them are by default available within the cluster, but not from the outside of the cluster or from the internet. This component uses iptables rules to expose the services using the host network.
c. As containers start and stop, scale up and down, there is a need to dynamically act on these changes and update various networking modules. This module makes sure conntrack entries are in sync, the ARP entries are accurate, and IPTables rules are in the right order.
d. Last but not the least, this component acts as the container runtime for the CNI interface and is responsible for invoking the appropriate CNI plugins to setup networking for containers.

In the case of imported clusters, this microservice doesn't act as the container runtime, and all the other network "sync" functionalities are disabled so as to not interfere with the already running networking plugin in the cluster.

2. **`cni-driver`**: This microservice is responsible for making various CNI plugins like IPSec, VLXAN pluggable into the cluster and sets up CNI network configuration files and binaries. The actual container is generic but it is deployed with a different CNI network config depending on the network plugin.
3. **`ipsec-agent`**: When IPSec networking solution is selected/installed in a cluster, this microservice is responsible for programming the IPSec tunnels, policies on the cluster nodes/hosts. It acts as an ARP proxy to enable encrypted cross host networking between the containers. This solution is available embedded clusters and is deployed as a Daemon Set.
4. **`vxlan-agent`**: When VXLAN networking solution is selected/installed in a cluster, this microservice is responsible for programming the VTEP interface, setting up routing on the cluster nodes/hosts. It acts as an ARP proxy to enable cross host networking between the containers using VXLAN tunnel. This solution is available embedded clusters and is deployed as a Daemon Set.

Non-Overlay Networking

Overlay networks are great for running containers in an isolated space. Features like host/published ports allow the services offered by containers to be consumed by external services. A shift is occurring in the network usage patterns where some users are envisioning moving the containers from the overlay network to the underlay network. Some Rancher users have an advanced architecture where they want to run the containers in the same network as the cluster nodes. To address user needs, Rancher now includes two new networking options: 1) "layer 3 routed" (per host subnet) and "layer 2 flat".

In the "layer 3 routed" solution, users can specify a subnet for the containers to run on each host. The routes to the subnet need to be programmed on the upstream switches or routers of the underlay network. For running this solution in AWS, Rancher installs the routes on the host directly or makes new route entries in the AWS VPC.

In the "layer 2 flat" architecture, the physical network interface of the host is placed on the same bridge where the containers are connected. This enables the containers to run directly on the underlay network, making them appear as regular machines connected to the network.

These two solutions are implemented using the CNI specification. Each of them would be a new catalog item available to install in Rancher environment. They would have their own agent containers similar to ipsec-agent or vxlan-agent.

network-manager and cni-driver containers are needed for any kind of CNI plugins to operate in Rancher environment.

High Availability

Rancher 2.0 is designed to function as a set of loosely coupled components without a single point of failure. The following explains how HA is assured in the following three major components of Rancher:

1. Rancher database
2. Rancher server
3. Embedded Kubernetes master

Rancher Database

Rancher 2.0 is designed to work with standard SQL databases such as MySQL. There are a number of ways to setup HA databases. Cloud providers offer Database-as-a-Service solutions such as the RDS service from AWS. Users can also setup HA databases themselves using Garela Cluster for MySQL or MySQL NDB Cluster.

Rancher Server

Rancher Server is a stateless component. Failure of an individual Rancher Server instance does not impact the functioning of the Rancher Server.

Kubernetes Master

Embedded Kubernetes Master consists of the multi-tenant API server, controller manager, and scheduler. All are stateless components and can withstand failures.

Loose Couplings between Rancher Server and Kubernetes Master

There are two points of coupling between Rancher Server and Kubernetes master:

1. The Netes-agent, itself a stateless component, proxies requests between Rancher Server and Kubernetes master. Failure of Netes-agent does not impact the Kubernetes master.
2. The auth proxy is a stateless service written in Go that performs authentication of all Kubernetes API calls. The auth proxy can withstand failures of individual components. A catastrophic failure of the entire auth proxy service will prevent the user from issuing API calls to the Kubernetes cluster, but will not impact the continuing functions of already-running Kubernetes workload.

Service Disruption during an Upgrade

Rancher upgrade starts with upgrading the server. After server upgrade completes, Rancher coordinates the upgrade of Rancher agents and embedded Kubernetes clusters.

Scalability

Scalability of Rancher Server

Because Rancher Agent is installed on all hosts, there is a limit on the total number of hosts a Rancher Server manage.Rancher creates database entries for clusters, environments, and containers. There is therefore a limit on these entities as well.

The following are the scalability goals of Rancher 2.0:

Entities	Scalability Limits
Clusters	1,000 Clusters
Hosts	1,000 hosts per cluster, 10,000 hosts across all clusters
Containers	30,000 containers per cluster, 300,000 containers across all clusters

Scalability of Embedded Kubernetes Master

The embedded Kubernetes cluster is subject to additional scalability requirements. Because Rancher runs one multi-tenant API server for all embedded Kubernetes clusters, there is a limit on the total number of hosts and containers across all embedded Kubernetes clusters.

The following are the scalability goals of the embedded Kubernetes master:

Entities	Scalability Limits
Embedded Kubernetes clusters	100 Clusters
Embedded Kubernetes Hosts	500 hosts per cluster, 1,000 hosts across all clusters
Embedded Kubernetes Containers	15,000 containers per cluster, 30,000 containers across all clusters

Get started with Rancher 2.0

To get started with Rancher 2.0, follow these two easy steps:

Step 1: Prepare a Linux Host

Rancher requires a single host installed with either Ubuntu 16.04 (kernel v3.1 O+) or RHEL/CentOS 7.3 as well as at least 2GB of memory, 80GB of local disk and the latest version of Docker.

Step 2: Start the server

To install and run Rancher server, execute the following Docker command on your host:

```
sudo docker run -d --restart=u n less-stopped -p 8080:8080 rancher/server:latest
```

It only takes a few minutes for the Rancher server to start up. The Rancher user interface is published on port 8080 by default, and can be accessed at http://<host_ip>:8080. Once Rancher has successfully been installed, the user interface will guide you through adding your first compute node and container.