

HobbyFarm

A Kubernetes-based, in-browser training tool

Agenda

- Introduction
- History and Need
- Design Goals
- Technology Choices & Architecture
- Backend Design
- Frontend Design
- Demo!
- Q&A

Introduction

Eamon Bauman

Senior Field Engineer

SUSE (formerly Rancher Labs)



Chris Kim

Software Engineer

SUSE (formerly Rancher Labs)



History - Rancher Rodeos

- In-person (pre-COVID) half-day seminars
- Introduction to
 - Docker
 - Kubernetes
 - Rancher
- Hosted at various worldwide locations
- Operated by Rancher Field Engineers



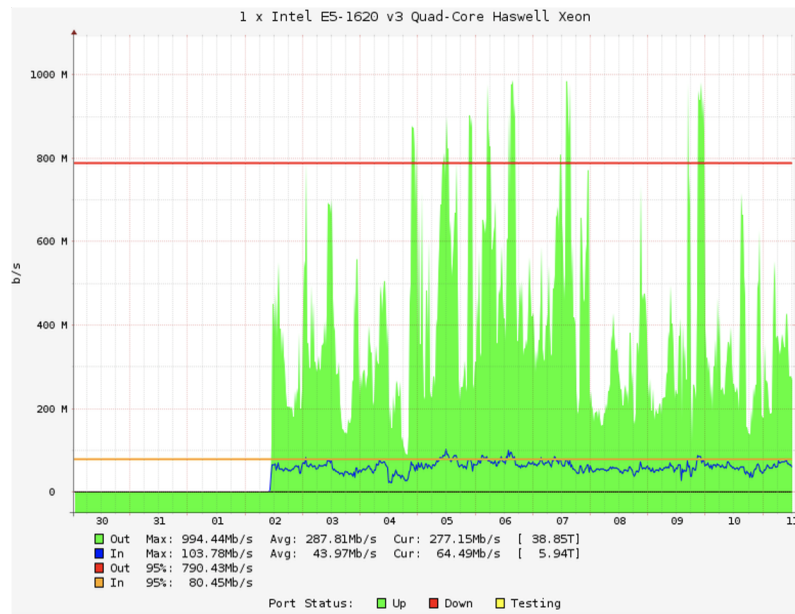
History - Rancher Rodeos

- Format was slide deck + interactive learning session
- Typically hosted at a hotel or conf. center
- Heavy use of Hashicorp Vagrant
- ...see the problem?



History - Rancher Rodeos

- Conference WiFi
 - Great for simple access
 - Not so great for large downloads
- Ubuntu 16.04 LTS: 291MB
 - ... times 30-50 users ...
 - ... all at once.
- Provisioning took *hours*
- Other issues
 - Lack of admin rights
 - Firewalls
 - ZSCALER



History - Rancher Rodeos

Potential Solutions

- **Pre-seed or use smaller images**
 - Doesn't solve for lack of admin rights
- **Cloud host VMs**
 - Largely a manual process
 - Also doesn't solve firewall (maybe no outbound tcp/22, or proxies in the way)
- **Something like Katacoda?**
 - "Magic Proxy" issues, due to no host header sent
 - Lack of flexibility

We needed something new.

Introducing HobbyFarm



HobbyFarm

- HobbyFarm is an interactive, browser-based, Kubernetes-powered learning tool
- HobbyFarm provides individual virtual machine(s) to users, along with instructions for learning new technologies
- HobbyFarm is 100% open source (<https://github.com/hobbyfarm>)

Primary Design Goals

- Entirely browser-based
 - Content written as markdown, rendered as HTML
 - Web shells connect into virtual machines
 - Even with restrictive firewalls, tcp/80 and tcp/443 to Internet and most* domains should be allowed
- Automated provisioning of resources (VMs)
- Schedule events ahead of time
 - VMs prepared prior to event
- Flexibility in infrastructure choices
 - AWS, Azure, VMware, etc.
- Flexibility in VM choices
 - Stateful capabilities, Host OS, resources, package installation, etc.

Secondary Design Goals

- “Click-to-run” functionality
 - Click code block in browser, executed in web shell
 - This quickly became a primary design *requirement*
- Full admin interface
 - Currently a work-in-progress, most operations available
 - Heavier configuration done “behind the scenes”
- Exportable content
 - Users can take the content with them for viewing later

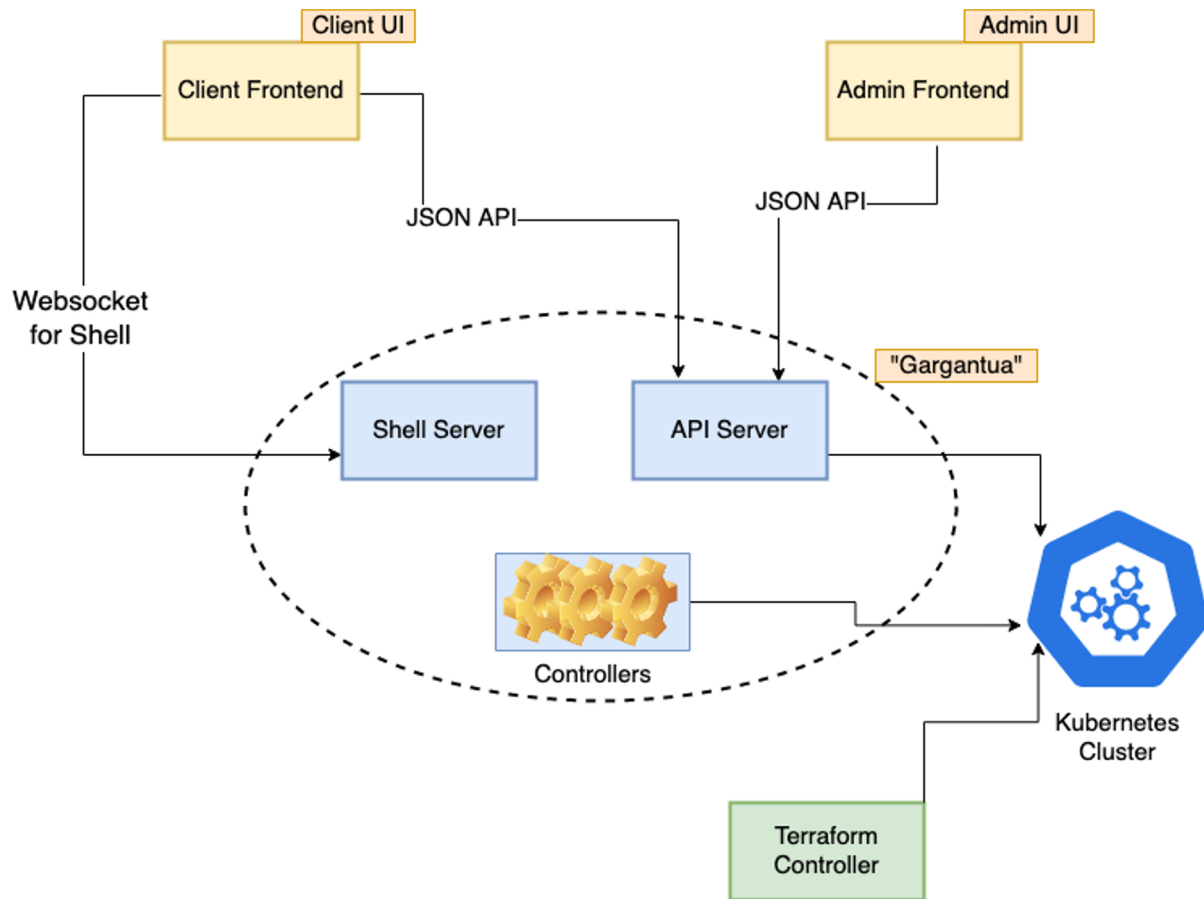
Technology Choices

- Go
- Kubernetes
 - Controllers are critical to this design
- Terraform
 - <https://github.com/rancher/terraform-controller>
- Angular
- Clarity Design System
 - <https://clarity.design/>

Virtual Machine Technology Choice

- Every user needs their own segmented space to work with
 - Containers did not provide enough isolation between users
- Rancher does not have a very large internal datacenter, so we would rely on the public cloud
- We analyzed the idea of using something like KubeVirt, i.e. running the VMs in containers
- We settled on using another Rancher project - the terraform-controller
- Even so, HobbyFarm is built to be pluggable with any provisioning tool (and you can even provision VMs out of band)

Architecture



Gargantua

- The HobbyFarm backend is a Golang monolith
- The name is inspired by the movie “Interstellar”



Backend Design

- Kubernetes client-go
- Kubernetes controllers
- JSON API
 - JWT
- Terraform Controller
- WebSocket Shell

Data Storage

- All metadata is stored in the Kubernetes cluster as custom resources (CR)
- The API Server and Shell Server both interact with the Kubernetes API to retrieve/store
- There are also a set of controllers (more on this later) that are constantly watching the Kubernetes API and acting on changes, which is the magic behind what makes HobbyFarm work

Why Kubernetes?

- **Controllers play a huge role**
 - HF leverages control loops and constant reconciliation
 - Client-go provides a great framework for writing reactive controllers
 - K8s controllers are a natural fit for that sort of work
- **Rancher is a Kubernetes company**
 - Dogfooding
 - Good experience

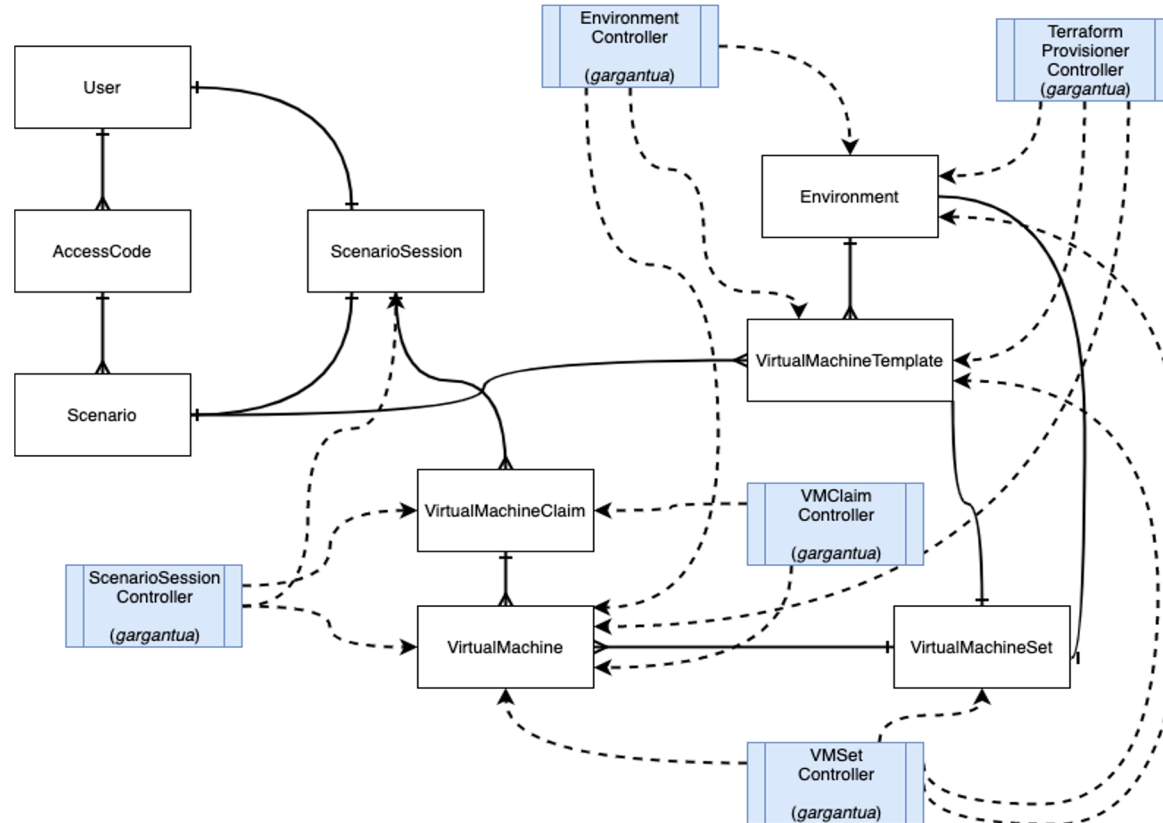
JSON API

- Gargantua serves a JSON-formatted REST API to the front end
- Every object that is served from the REST API has a corresponding

Kubernetes Custom Resource

- However, not every HobbyFarm CR is served via the REST API

Software Entities



Software Entities - VMs & Environments

- Environment
- VirtualMachineTemplate
- VirtualMachine
- VirtualMachineClaim
- VirtualMachineSet
- DynamicBindRequest & DynamicBindConfiguration

Software Entities - Content & Access

- Scenario
- Course
- AccessCode
- ScheduledEvent
- User

Basic K8s Controller Architecture

- Two distinctive “loops” in a controller
- The basic controller architecture of Gargantua operates as follows
 - The controller performs a K8s API Watch for changes to the object that the controller operates upon
 - When a change is seen, the object name/metadata is then queued into a workqueue
 - The controller in another loop is constantly popping objects off of the workqueue
 - When it pops an object off the workqueue, it operates on the object
- You can store (pretty much) anything in a Custom Resource
- This makes using the K8s API/client-go very powerful
- Note that there can be many loops watching the same workqueue

Gargantua Controller Example - Scheduled Event

- Read in list of scheduled events and determine if action required
 - Brand new event
 - Start or end time passed
- If action required, reconcile to desired state
 - Create access code
 - Create VM set
 - Trigger VM provisioning

Virtual Machines

- Virtual Machines metadata is stored in HobbyFarm/K8s API
- The design is supposed to be agnostic, as to allow any type of virtual machine
 - Amazon EC2
 - Azure
 - VMware
 - KubeVirt
 - Etc.
- Provisioning is handled outside of HobbyFarm
 - There are data structures to represent a “desired” VM and an external provisioning tool can actually create the VM using these
 - Gargantua has a terraform-controller integration

rancher/terraform-controller

- rancher/terraform-controller is another K8s-powered Golang tool
- It performs terraform apply based on the K8s CR definition
 - Originally built for Rancher to serve as a general purpose VM provisioner
- Uses the rancher/wrangler framework for controller management
 - This is different from HobbyFarm, which does not use a framework
- Fundamentally, the ideas are the same as HobbyFarm

Shell Server

- Takes an incoming websocket connection from the user-facing frontend
- Establishes an SSH session with the user's desired VM
- Acts as a middleware to handle incoming websocket messages and relay them to the open tty
- `io.pipe`
 - Incoming websocket messages are piped from the socket to the shell's `stdin`
 - Outbound messages from `stdout`, `stderr` are piped to the socket's `write`

Frontend Design

- Angular
- Auth0 JWT
- Clarity Design Framework
 - Themed
- Ngx-markdown
- xterm
- Hacky dynamic HTML insertion
- Click to run functionality

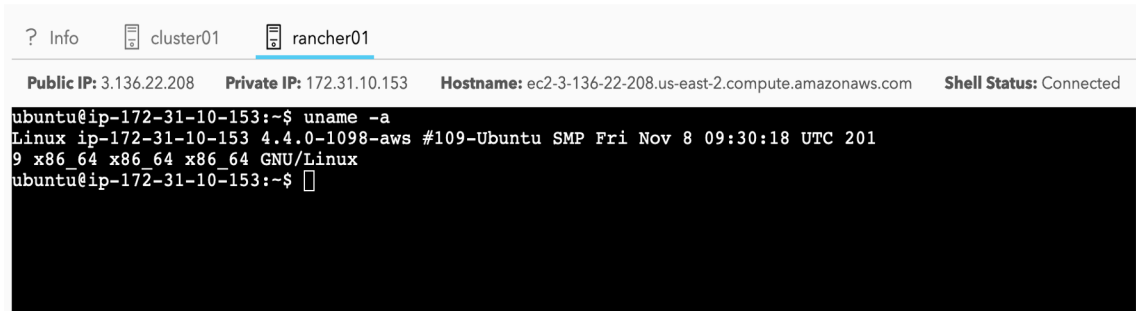
Angular + Clarity Design Framework

- I/we are **not** great wranglers of css nor design
- Did not want to be hampered by learning and designing an entire frontend
- But still needed a nice looking UI
- Clarity Design Framework made this trivial
 - Opinionated
 - Built in Angular from the ground up
 - Looks really nice with very little effort



In-Browser Terminal

- According to a user's VMClaim, one or more tabs are created on the page
- Each tab contains an xterm.js instance
- Each tab is an Angular component that calls the Shell Server and hooks up a websocket to the xterm.js instance
- User gets a **real** terminal, not emulated. It is direct input/output (via shell server piping) from their VMs
- All control operators work, e.c. ^C



```
? Info cluster01 rancher01
Public IP: 3.136.22.208 Private IP: 172.31.10.153 Hostname: ec2-3-136-22-208.us-east-2.compute.amazonaws.com Shell Status: Connected
ubuntu@ip-172-31-10-153:~$ uname -a
Linux ip-172-31-10-153 4.4.0-1098-aws #109-Ubuntu SMP Fri Nov 8 09:30:18 UTC 201
9 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ip-172-31-10-153:~$
```

Markdown Rendering

- Needed ability to insert variables into rendered markdown
- Variable values not known until runtime
 - Data from the VMs, e.g. public IP, hostname, etc.
- *Ngx-markdown* custom renderer
 - Custom function to handle rendering of markdown code blocks
 - Looks for `${vminfo:x:y}` code where x is a VM and y is a property on that VM
 - Example: `${vminfo:machine01:private_ip}` would be replaced with **172.16.34.55**
 - Sourced from a VM object obtained via API calls, passing user info and context

RKE uses SSH tunneling, which is why we generated the keypair in the first part of this scenario.

```
cat << EOF > rancher-cluster.yml
nodes:
  - address: 3.136.22.208
    internal_address: 172.31.10.153
    user: ubuntu
    role: [controlplane,etcd,worker]
addon_job_timeout: 120
EOF
```

[^ Click to run on Rancher01](#)

Click-to-Run

- Provide ability for user to click on a code block and insert into terminal
- Leverages same ngx-markdown hook as variable rendering
 - Looks for ``ctr:machine01
 - Any code in that block becomes the code to run in the terminal
- Each terminal component in Angular listens to an Observable of “CTR messages”
- If a CTR message matches that terminal’s machine id, inserts content into terminal

The following command will generate the keypair and copy it into the file.

```
ssh-keygen -b 2048 -t rsa -f \  
/home/ubuntu/.ssh/id_rsa -N ""  
cat /home/ubuntu/.ssh/id_rsa.pub \  
>> /home/ubuntu/.ssh/authorized_keys
```

^ Click to run on Rancher01

Dynamic HTML

- Angular does *not* like dynamic HTML
 - Components and their templates are expected to be static
- You can substitute *components* dynamically
 - But not the content *of* the component
- HobbyFarm's markdown must be dynamic HTML
 - We can't generate a component for every step in a scenario
 - We can't just string insert the markdown onto the page - need to build tags based on markdown

Dynamic HTML

- HobbyFarm has a dynamic HTML component that...
- ... takes input HTML
- “Sanitizes” it
- Sets it as the innerHTML of a the containing Angular object (e.g. `<dynamic-html [content]="content"></dynamic-html>`)
 - the child elements of that component tag are the rendered-from-markdown HTML elements, e.g. `<p>`, ``, `<code>`, `<h1>`

Problems Faced

- Terraform is not always reliable
- There is abstraction upon abstraction upon abstraction
- K8s API is not fast
 - K3s + MySQL solves this somewhat, but K8s API was still not designed for this
 - We looked at using an embedded K3s server for the sole purpose of HobbyFarm data storage
- Users click instructions too many times
- User proxies

**Demo
Time!!**