Revamping a legacy backend

by Umar Nizamani

- Existed for over 7 years
- Started with IoT and pivoted over the years to build various health care projects
- 2 year ago ready to build a new product
- Run legacy backend for at-least 2 more years
- A total team of ~25 engineers

Working in Health Care

Health Care Data

- Working with sensitive user data
- Certified for major information security standards (ISO 27001 & NEN 7510)
- Data has to be kept within the Netherlands

NICEDAY

- Realtime chat & calling for mental health treatment

- Used by mental health professionals as their core tool for treating clients

- Different product on web + mobile

Maintaining the legacy

The legacy backend

- Several micro-services that rely on each other
- Servers setup on various different providers
- All servers created manually by different engineers
- Only 1 senior engineer truly understood the architecture

Technical Debt

- Deployments were dreaded
- Developers were afraid to fix core problems
- Downtime usually meant calling the senior engineer
- On-boarding new engineers was mainly watching the senior engineer solve problems

Familiar?

A common startup problem

- As startups pivot and scale the backend suffers
- Technical debt also applies to servers
- Time to do it manually maybe < time to automate it

But how do we fix it?

The ideal picture

- The entire environment has to be reproducible
- Developers should not spend time on tasks that are easy to automate
- Everything and everyone should be redundant

Lets make it happen

Step 1

Containers

Containerise the stack

- Packaged all services in containers
- A simple bash file to initiate each service
- docker-compose for the whole backend

Containerise the stack

Microservice #1

```
while (wait for required services)
  sleep;

initialise microservice #1
```

docker-compose

- Simplifies developer on boarding
- Easy to see all dependencies for each service
- Reproducible backend
- Encourages all developers to contribute

```
$> docker-compose up
```

Whats the big win?

- Major boost for development workflow
- Simplifies setting up a local dev environment
- Run the entire backend locally to test end to end

Step 2

Infrastructure as Code - 1

Terraform to create servers

- Created new servers using Terraform
- Frees us from vendor lock-in
- Server, firewall and network config all in Git
- Pull Requests to approve new servers

Terraform config

```
resource "openstack_compute_instance_v2" "mariadb_staging_1" {
  name              = "mariadb-staging-1"
  image_id          = "3bf30bab-8afe-241b-a3bf-d2f98aae7237"
  flavor_name       = "Standard 2"
  key_pair          = "server_key"
  security_groups = ["default",
    “${openstack_compute_secgroup_v2.mariadb_exporter.name}”
    ,… ]
  availability_zone = "NL1"

  network {
    name = "staging"
  }
}
```

Explaining the config

```
resource "openstack_compute_instance_v2" "mariadb_staging_1" {
  name            = "mariadb-staging-1"
  image_id        = "3bf30bab-8afe-241b-a3bf-d2f98aae7237"
  flavor_name     = "Standard 2"
  key_pair        = "server_key"
  security_groups = ["default",
    “${openstack_compute_secgroup_v2.mariadb_exporter.name}”
    ,… ]
  availability_zone = "NL1"

  network {
    name = "staging"
  }
}
```

Specifying an OpenStack compute instance

Explaining the config

```
resource "openstack_compute_instance_v2" "mariadb_staging_1" {
  name              = "mariadb-staging-1"
  image_id          = "3bf30bab-8afe-241b-a3bf-d2f98aae7237"
  flavor_name       = "Standard 2"
  key_pair          = "server_key"
  security_groups = ["default",
    “${openstack_compute_secgroup_v2.mariadb_exporter.name}”
    ,… ]
  availability_zone = "NL1"

  network {
    name = "staging"
  }
}
```

flavor_name is the server spec name given by the provider (e.g Standard 2 = 2 Cores, 2 GB RAM, 32 GB HDD)

Explaining the config

```
resource "openstack_compute_instance_v2" "mariadb_staging_1" {
  name              = "mariadb-staging-1"
  image_id          = "3bf30bab-8afe-241b-a3bf-d2f98aae7237"
  flavor_name       = "Standard 2"
  key_pair          = "server_key"
  security_groups = ["default",
    “${openstack_compute_secgroup_v2.mariadb_exporter.name}”
    ,… ]
  availability_zone = "NL1"

  network {
    name = "staging"
  }
}
```

security_groups specify the firewall rules for this server

Explaining the config

```
resource "openstack_compute_instance_v2" "mariadb_staging_1" {
  name              = "mariadb-staging-1"
  image_id          = "3bf30bab-8afe-241b-a3bf-d2f98aae7237"
  flavor_name       = "Standard 2"
  key_pair          = "server_key"
  security_groups = ["default",
    “${openstack_compute_secgroup_v2.mariadb_exporter.name}”
    ,… ]
  availability_zone = "NL1"

  network {
    name = "staging"
  }
}
```

availability_zone the data center to setup this server in

Controlling usage

- Store the current state of the servers in S3
- Clear guideline for using terraform

```
$> terraform plan
$> … 👀
$> terraform apply
```

Whats the big win?

- Free from provider lock-in
- All firewall rules clearly documented
- Very easy to setup clusters

Step 3

Infrastructure as Code - 2

Ansible to provision servers

- All server software installed using Ansible
- Using **roles** to make sure all servers have same core setup
- Extremely easy to setup clusters

Ansible config

```yaml
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

Explaining the config

```yaml
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

hosts specify the group of servers this config should be applied to

Explaining the config

```
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

{{ env }} parameter filled with staging/production depending on what server group is targeted

Explaining the config

```
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

encrypted folder with all files encrypted using ansible-vault, password shared via other mediums

Explaining the config

```
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

Open Source roles found on GitHub that setup a MariaDB Galera cluster

Explaining the config

```
- hosts: mariadb
  become: true
  vars_files:
    - "encrypted/{{ env }}/mariadb"
  vars:
    mysql_secure_files: "encrypted/{{ env }}/key.enc"
  roles:
    - mrlesmithjr.mariadb-galera-cluster
    - mariadb-secure
    - role: internal-backup-role
      tags:
        - backup
    - prometheus-mysqld-exporter
```

internal-backup-role an in-house role used to backup all supported databases in a unified way

Whats the big win?

- Clusters made effortless
- Fully reproducible servers managed in Git
- Ensure same configuration between Staging/Prod
- Easy to incrementally improve entire infrastructure
- ansible-vault to manage your secrets

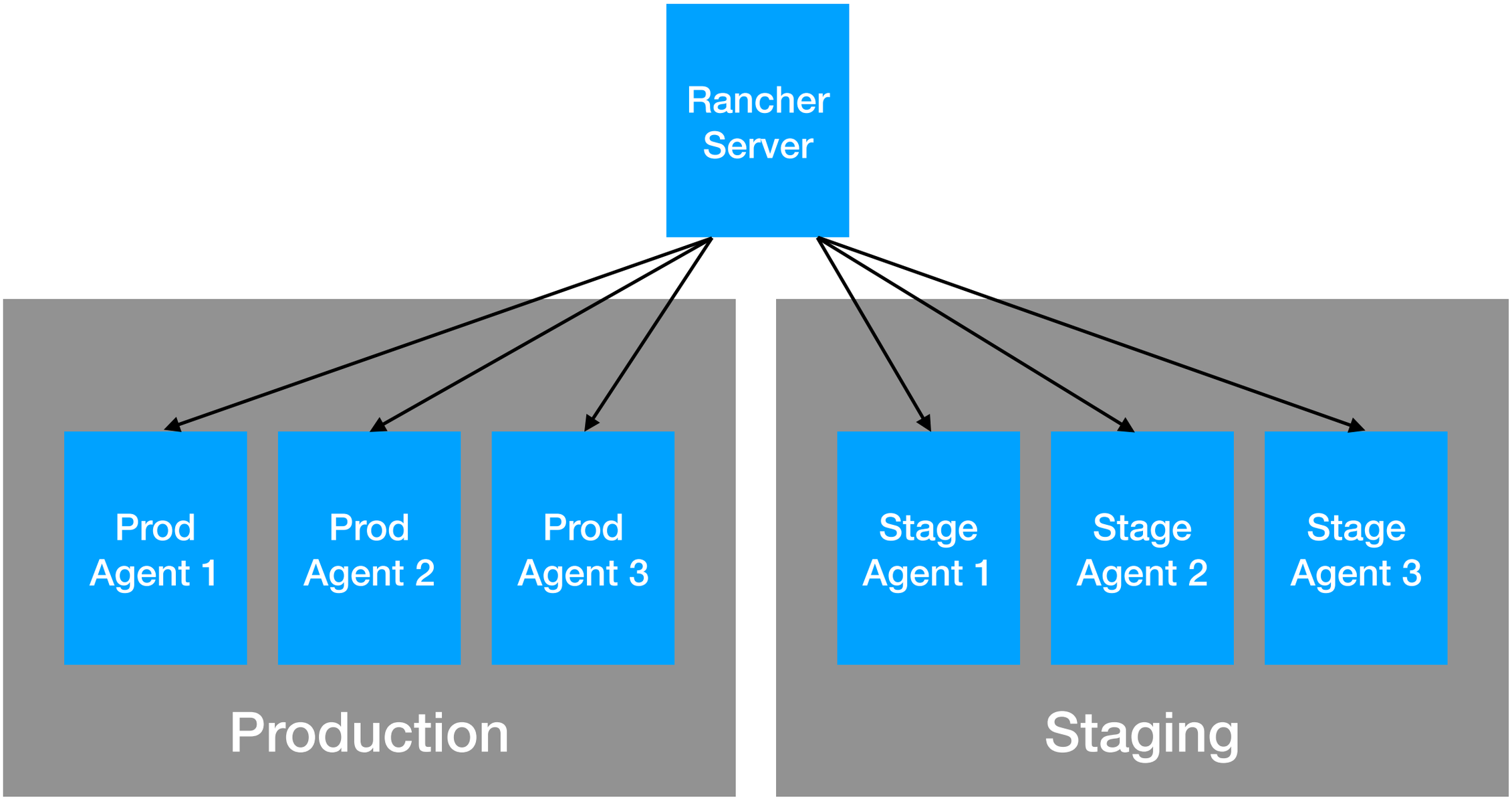
Step 4

Orchestrating containers

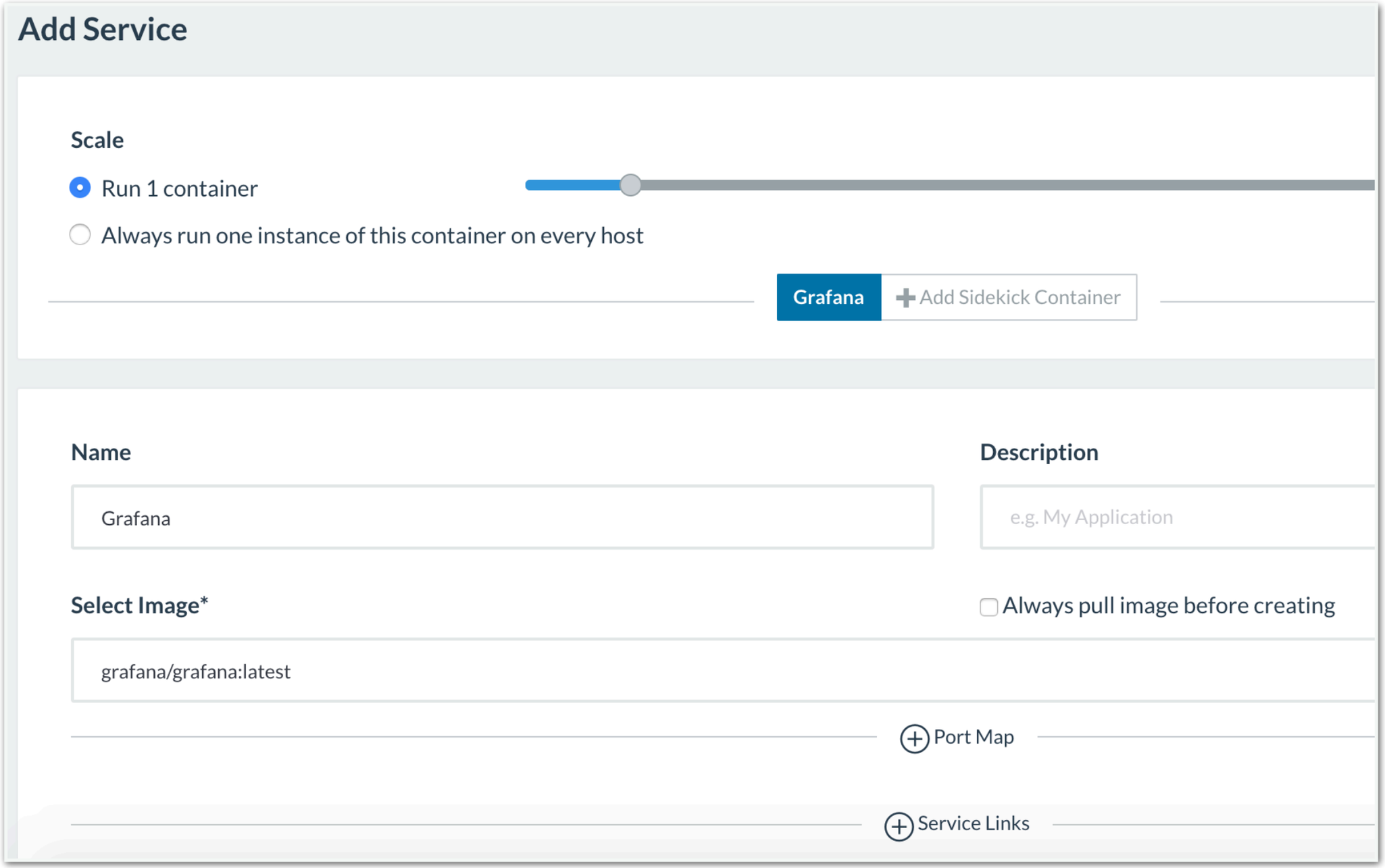
Rancher

- A UI for setting up container orchestration
- Packed with tools to ease deployment
- Allows non infrastructure team to setup containers
- Now based on Kubernetes

Rancher Environments



Creating a service



Health Checks

Command | Volumes | Networking | Security/Host | Secrets | Health Check | Labels | Scheduling

Type ○ None ○ TCP Connection Opens ● HTTP Responds 2xx/3xx

Port* 3000

HTTP Request* GET | /login | HTTP/1.0

Initializing Timeout 60000 ms

Reinitializing Timeout 60000 ms

Check Interval 2000 ms

Check Timeout 2000 ms

Healthy After 2 successes

Unhealthy After 3 failures

When Unhealthy

○ Take no action

● Re-create

○ Re-create, only when at least 1 container is healthy

Load Balancer

Name

Load Balancer

Description

e.g. Balancer for mycompany.com

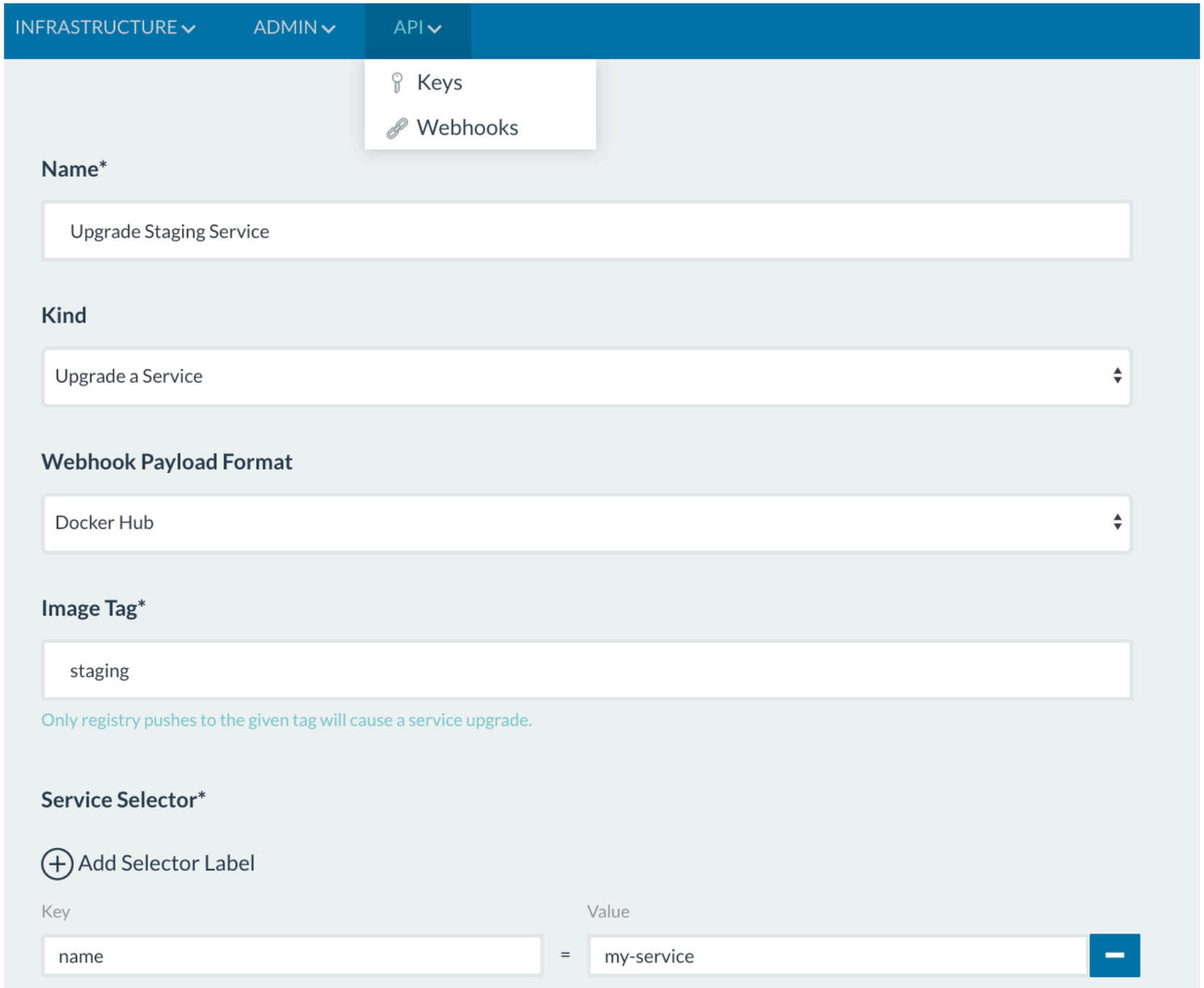
Port Rules ⊕ Add Service Rule ⊕ Add Selector Rule

Access*	Protocol*	Request Host	Port*	Path	Target*	Port*
Public	HTTPS	grafana.mywebsite.com	443	e.g. /foo	Grafana/grafana	3000

Host and Path rules are matched top-to-bottom in the order shown. Backends will be named randomly by default; to customize the generated backends, provide a name and then refer to that in the custom haproxy.cfg. Show custom backend names. Show host IP address options.

Built-in HAProxy load balancer that can connect to services

Web hooks


INFRASTRUCTURE
ADMIN
API
Keys
Webhooks
Name*
Upgrade Staging Service
Kind
Upgrade a Service
Webhook Payload Format
Docker Hub
Image Tag*
staging
Only registry pushes to the given tag will cause a service upgrade.
Service Selector*
Add Selector Label
Key
Value
name
=
my-service

Whats the big win?

- Easy to manage clusters for your services
- Built in load balancer and health checks
- UI with access control to allow developers to deploy test services
- Ready for continuous deployment

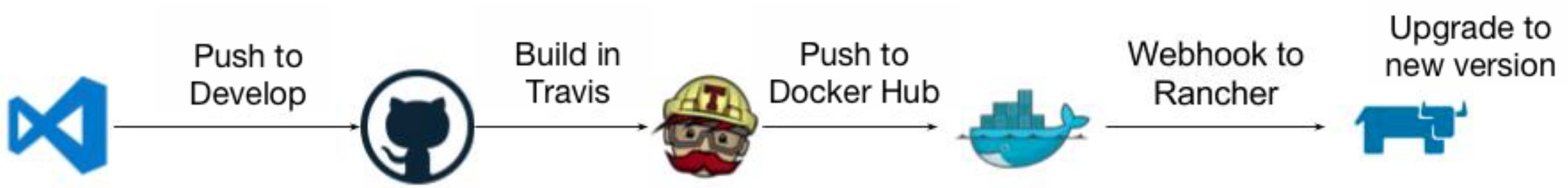
Step 5

Continuous Deployment

Travis + Docker Hub

- Each commit runs tests and reports code quality
- Commits to develop push new image to docker hub
- Dockerhub fires web hook to update Rancher container
- Health checks + rollbacks for bad commits
- docker-compose allows running entire backend in Travis to run integration tests
- Use Travis to notify the team about releases

Travis + Docker Hub



Whats the big win?

- Developers don’t need to touch infrastructure
- QA notified for release otherwise one click rollback
- Bots check code quality
- Each release tagged and stored in docker hub for easy rollback to any version

Step 6

Gearing up for Production

End to end health checks

- Created status endpoints that queries all services it depends on
- Container health checks for avoiding failures
- Return response times for each internal query

```
{"service-1": "0.016ms", "service-2": "0.01ms", "mysql": “0.006ms” …}
```

Configuration Management

- Services relied on configuration files and lot of work to move to any other format
- Stored all configuration files encrypted in Ansible
- Bash script to hot reload service on config change

```
inotifywait -m -e modify /etc/service/service.conf | while read events;
do supervisorctl restart service; done
```

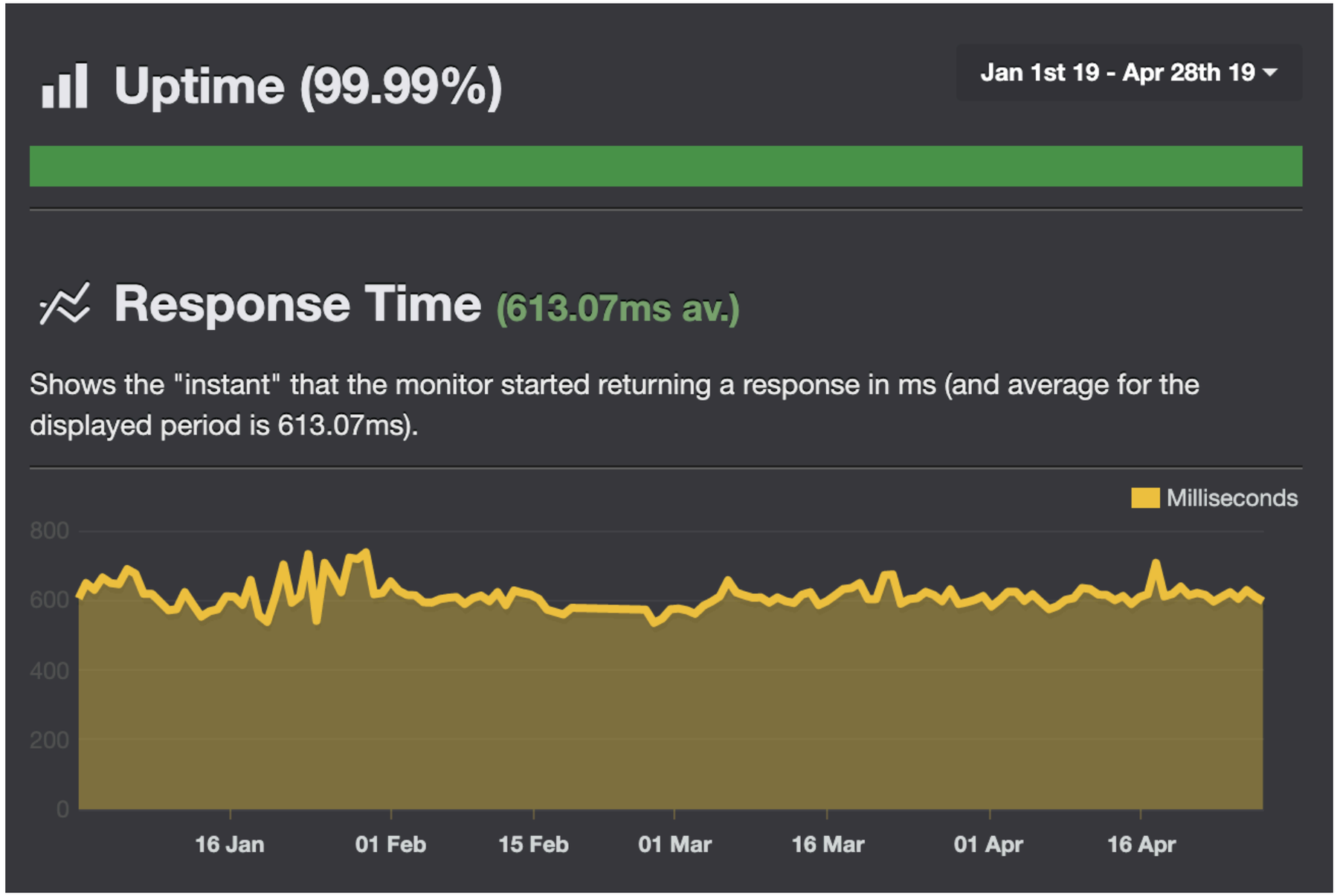
Incident Management

- Prometheus to monitor the status of all services
- Alert Manager to configure severity of incidents
- Incidents and alerts reported to Ops Genie
- On Call schedule in Ops Genie for the team
- Severity based alert (Push Notification, SMS, Call)

External Health Checks

- Uptime Robot to ping externally visible servers
- Health checks performed from multiple locations
- A status page to share status with non-tech team
- Monitor provider performance and uptime

The results



Total 16 minutes of downtime in 2019

8 minutes because of our provider

Maintenance

- All servers are now clustered so almost never a critical failure

- Averse to hardware failure, provisioning new identical server takes ~ 20 minutes

- All tooling that runs on docker now supports 1 click upgrades (e.g Sentry, Kibana, Metabase, …)

A stronger company

- We allow backend engineers to learn infrastructure management by being on call

- Setup several new tools for improving internal development workflow

- Operations team is working on prevention not cure

Thank You