To Infinite Scale and Beyond

Operating Kubernetes Past the Steady State

I’m Austin

I work in Core Infrastructure at Spotify

We build the infra that powers your audio

I’m Jago

Eng Director, Kubernetes & GKE at Google

We build the infra that powers the infra that powers your audio

Goals for Today

1. Define and discuss why it’s important to “think beyond steady states”
2. Highlight the core operational challenges of running Kubernetes at scale
3. Share mitigation tactics and best practices

Defining steady states

Steady states are the *mostly* predictable milestones of scale that your platform manages.

Examples:

- Environment after initial provisioning - first deployment!
- Running workloads on a known set of underlying components
- Even high end of predictable curve - peak load or scalability at a point in time:
  - Game of Thrones episode launch (HBO @ Kubecon 2017)
  - Black Friday for many companies

They are the easy part…

Steady States:

1. Tend to be predictable and a single event; they happen once
2. Can often be managed by autoscaling (assuming HPA settings!)

In contrast, supporting a business through growth and evolution is a process - continuous until you brush up your resume..

But often, we focus too much on these steady states and overlook the operational challenges of running at scale

Challenges at Scale

- “There is no Cloud - it’s just someone else’s computer[s].”
  - Even someone else’s computers will fail… sporadically
- Kubernetes takes care of some issues
  - Pods and even dead VMs are automatically replaced
- But zones or even regions can go down

How do you seamlessly handle those failures?

How do you build a globally available service on a collection of demonstrably fallible zones & regions?

… by thinking beyond the steady states!

Takeaways

Things we hope you can take
and act on from this session

Automate Cluster Upgrades

Automate cluster upgrades for both minor upgrades and patch versions of Kubernetes and related extensions

Takeaways

Things we hope you can take
and act on from this session

Manage Multicluster Deployments

Create multi-tenant ephemeral clusters across
regions and manage cross-cluster deployments

Let’s Jump In

Takeaways

Things we hope you can take
and act on from this session

Automate Cluster Upgrades

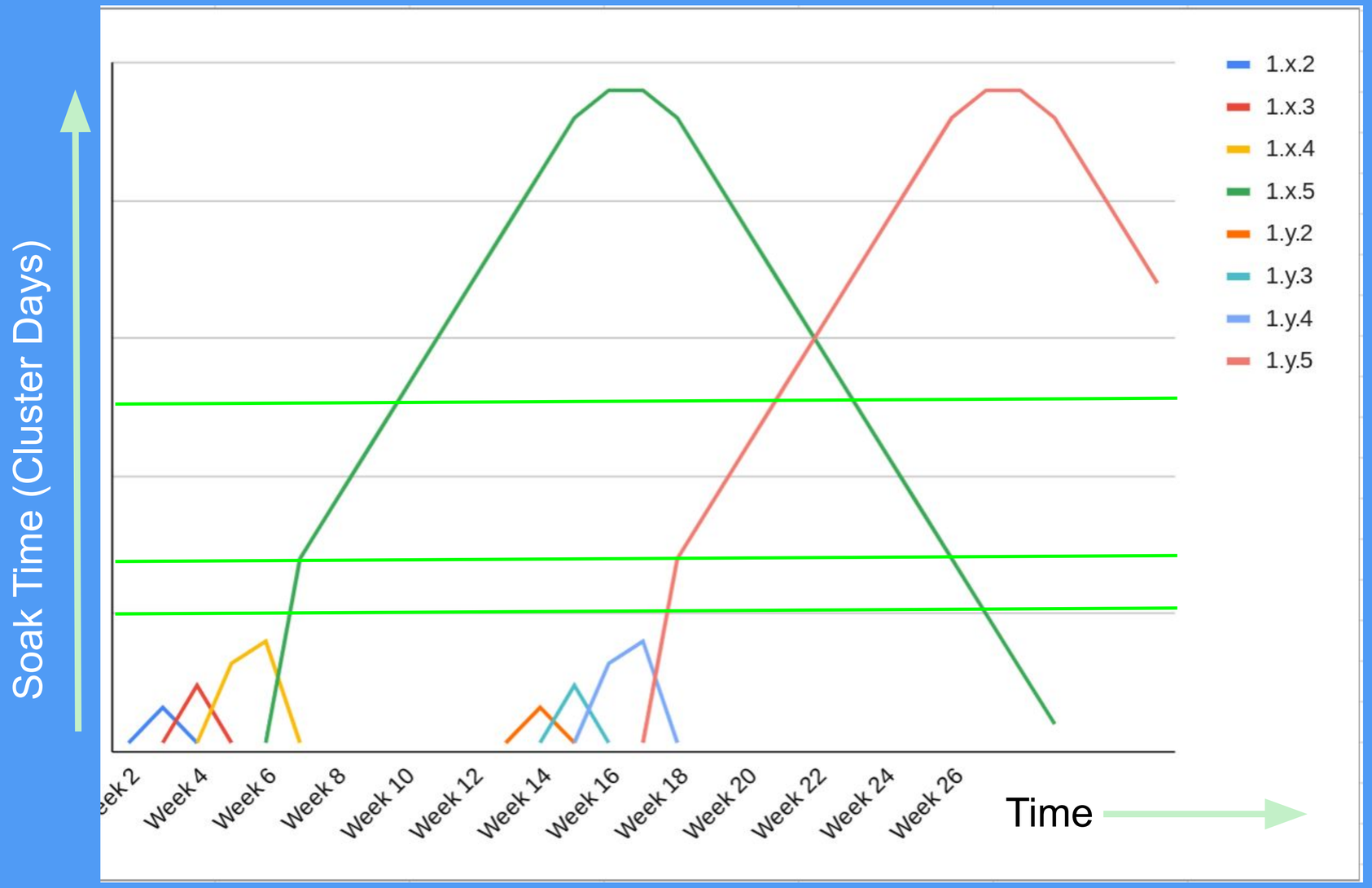
Automate cluster upgrades for both minor upgrades and patch versions of Kubernetes and related extensions

Kubernetes Minor Release Adoption - Idealized

New versions introduced into Alpha Clusters & Rapid Channel

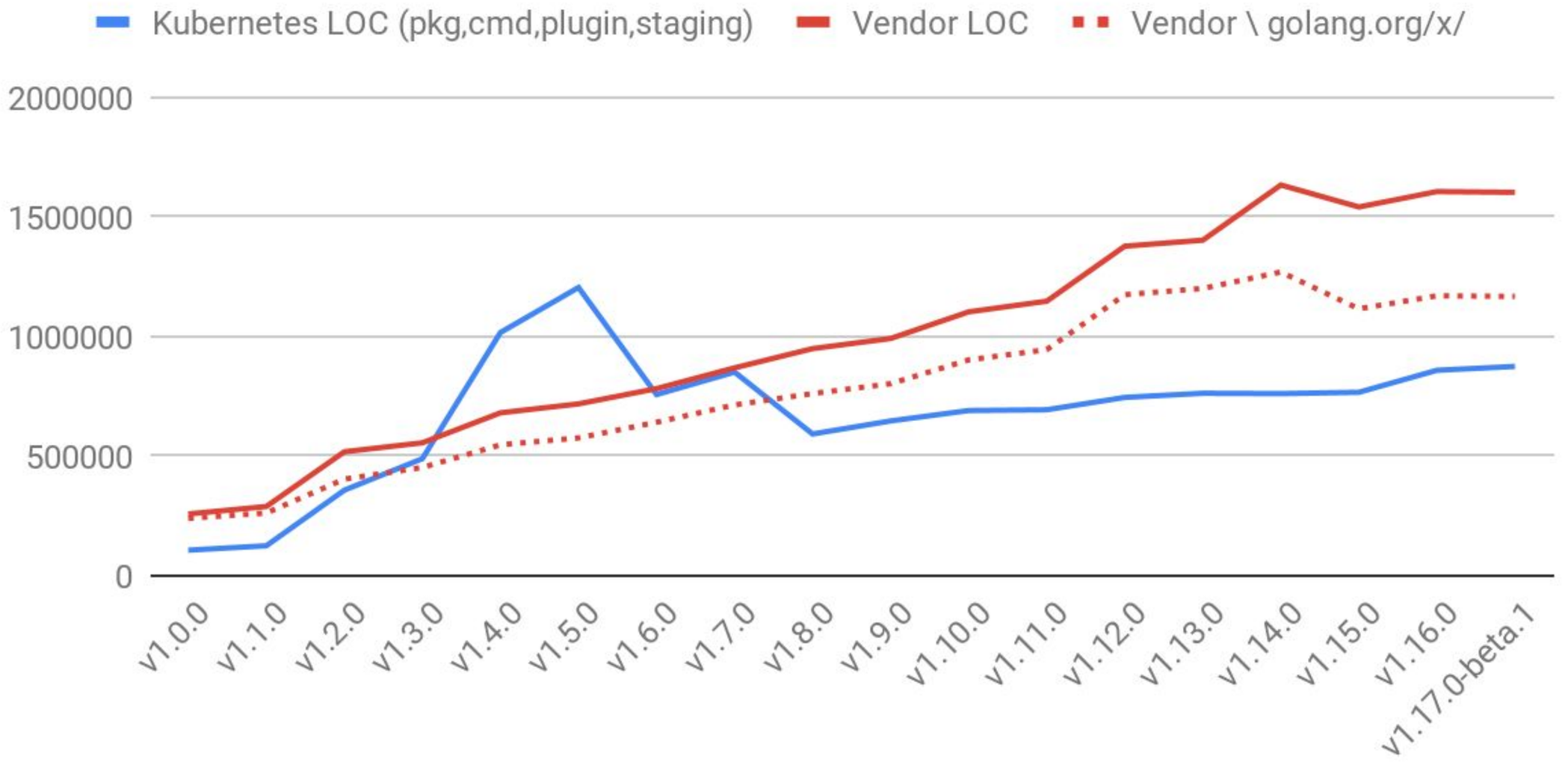
Versions that accumulate enough cluster days & meet SLOs are promoted:

1. To GA & available in Regular Channel
2. To default & upgrade target
3. To Stable Channel



Kubernetes Lines of Code

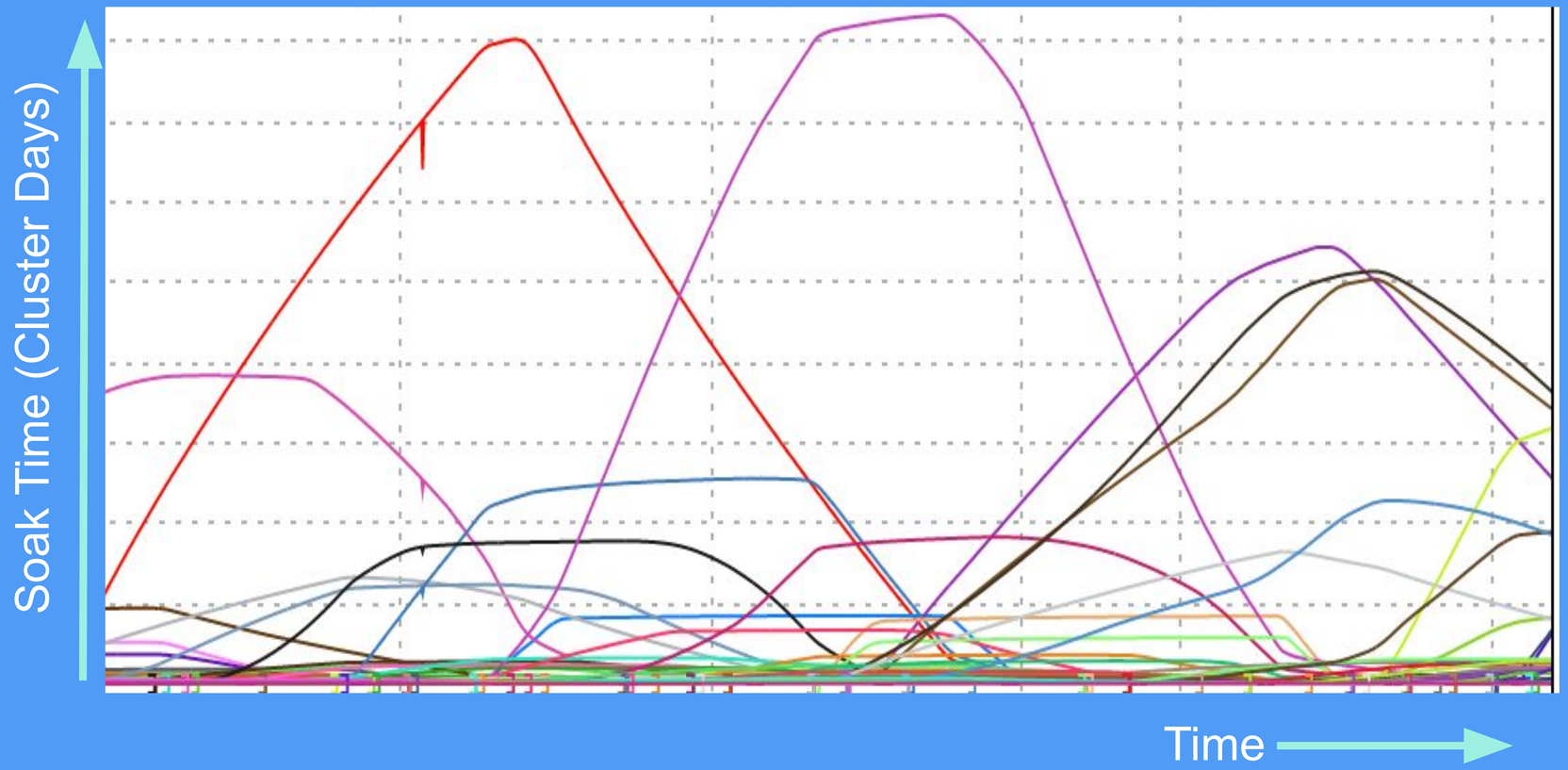
Non-blank Non-comment Non-test Go



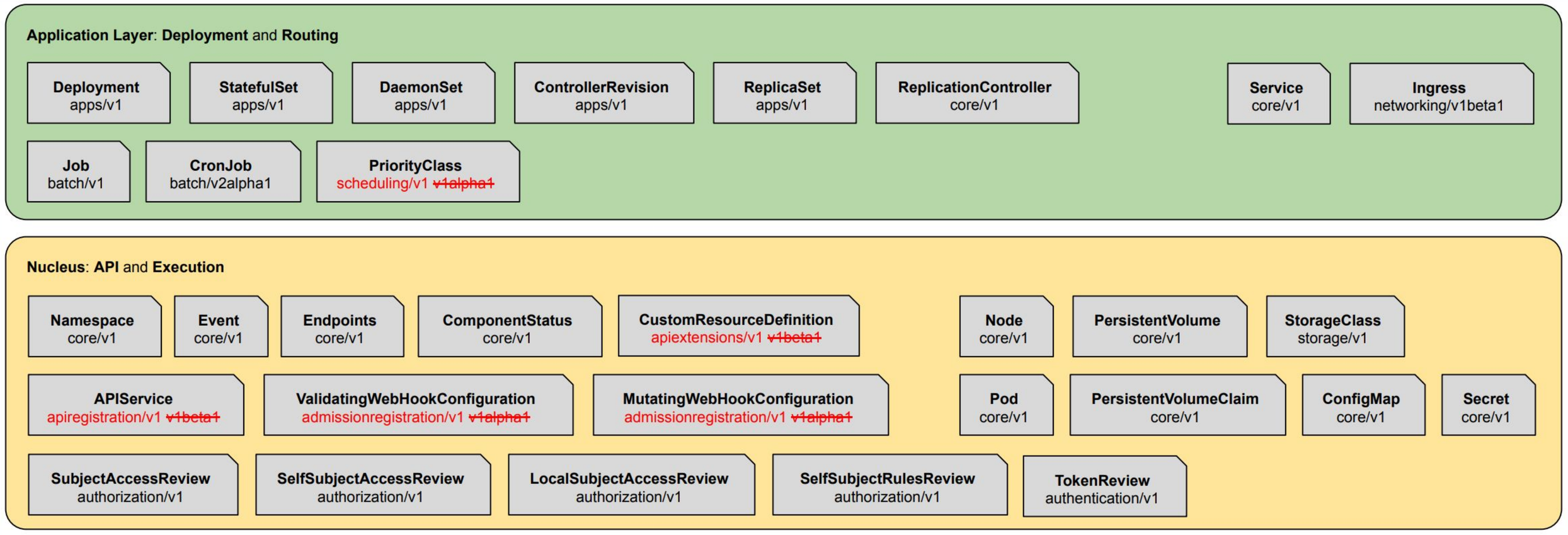
GKE Soak Time for Cluster Versions - Actual

In reality, upgrades happen much more frequently:

1. **Security Patches**
2. **Bug fixes/Regressions**
   a. **K8s Components**
   b. **Add-Ons**
   c. **Container Runtime**
   d. **Base OS**
3. **Novel customer use stresses in a new dimension.**



Some Good News



Some Good News

~All APIs in the Nucleus and Application layers are stable as of 1.17*.

    With stable APIs comes backwards compatibility guarantees, portability guarantees (Conformance Tests), and (often) scale testing.

Community alignment around supporting 12 months (4 minor releases)

    Users can choose to upgrade several versions sequentially

* Ingress and CronJob in progress.

Learnings

Automate Frequent, Error Prone Changes

- Cluster Upgrades happen *several times* per Minor release
- Even older, more stable versions receive backported bug fixes and security patches

Kubernetes V1 is (just about) Done!

- API Maturity has improved; rate of new functionality has decreased
- Upgrades are now about security and stability not getting the newest features fastest

Being too early, or too late, means you are more alone (read at Risk)

- Choose Release Channel or Minor Release based on Risk Tolerance
- Watch Release Notes, patch release PR / cherry pick rate to manage risk

Learnings (continued)

Avoiding upgrades leads to ‘haunted graveyards’

When something is scary, do it ***more*** not less

Productize and Limit Customization

Every customization that is not reproducible is debt

ssh + apt-get install = node that can’t be upgraded or replaced

Build a Culture of **dynamic stability**

Smaller, more frequent changes typically mean smaller, more contained incidents

Teams that release more create more resilient systems

Observability & Controlled Releases are necessary

Ok, sure, but....

HOW?

Solutions - basic

Select **Regular** Release Channel; Opt In to Node Auto Upgrade & Node Auto Provisioning; Enable Logging.

Solutions - intermediate

Create a **Canary Cluster** in a Day 1 Region {europe-west3, us-east1} and **Production Cluster(s)** in a Day 4 Region {asia-northeast2, asia-south1, europe-north1, europe-west4, us-central1}

Select **Regular** Release Channel; Opt In to Node Auto Upgrade & Node Auto Provisioning; Enable Logging.

Implement StackDriver Uptime Checks for a few key health indicators, and set up corresponding alerts.

If Uptime Check fails for Canary Cluster, use maintenance windows to delay upgrade to Production Cluster(s).

Solutions - advanced

Embrace Declarative Configuration - Configuration as ~~Code~~ *Data*

- Automate cluster creation, configuration & provisioning other cloud resources
  - Use Terraform* (Be careful! Read the Terraform Up & Running Book)
  - Use Kustomize for configuration overlay
  - Explore Config Connector to use Kubernetes Resource Model everywhere

Manage Upgrades of Large (1000+ Nodes) Nodepools

- Surge upgrades; or
- Create new Nodepool at the new version & migrate traffic to it

Invest in a Global Topology Strategy

- Multi-cluster, HA & Disaster Recovery
- Note: Networking gets complicated

Takeaways

Things we hope you can take
and act on from this session

Manage Multicluster Deployments

Create multi-tenant ephemeral clusters across
regions and manage cross-cluster deployments

Challenges

1. Independent of what's possible for your scale, you don't want one big cluster
   - **Assume failure! It will happen! One cluster is a single point of failure!**
     - The scheduler could fail to assign any new pods to nodes
     - The cluster autoscaler could fail to increase the size of the cluster to meet demands
     - Add-ons, CRDs/Operators, 1st or 3rd Party Applications can DDoS the API Server
     - A Terraform mistake can delete an entire cluster
       - KubeCon Europe 2019: How Spotify Accidentally Deleted All its Kube Clusters

2. Deviation in your cluster configuration is really bad
   - During an incident, it should be trivial to create or replace a cluster

Learnings

Insights

Developers don’t care which cluster they deploy to, why expose it to them front and center?

Network is complex… and highly challenging to evolve

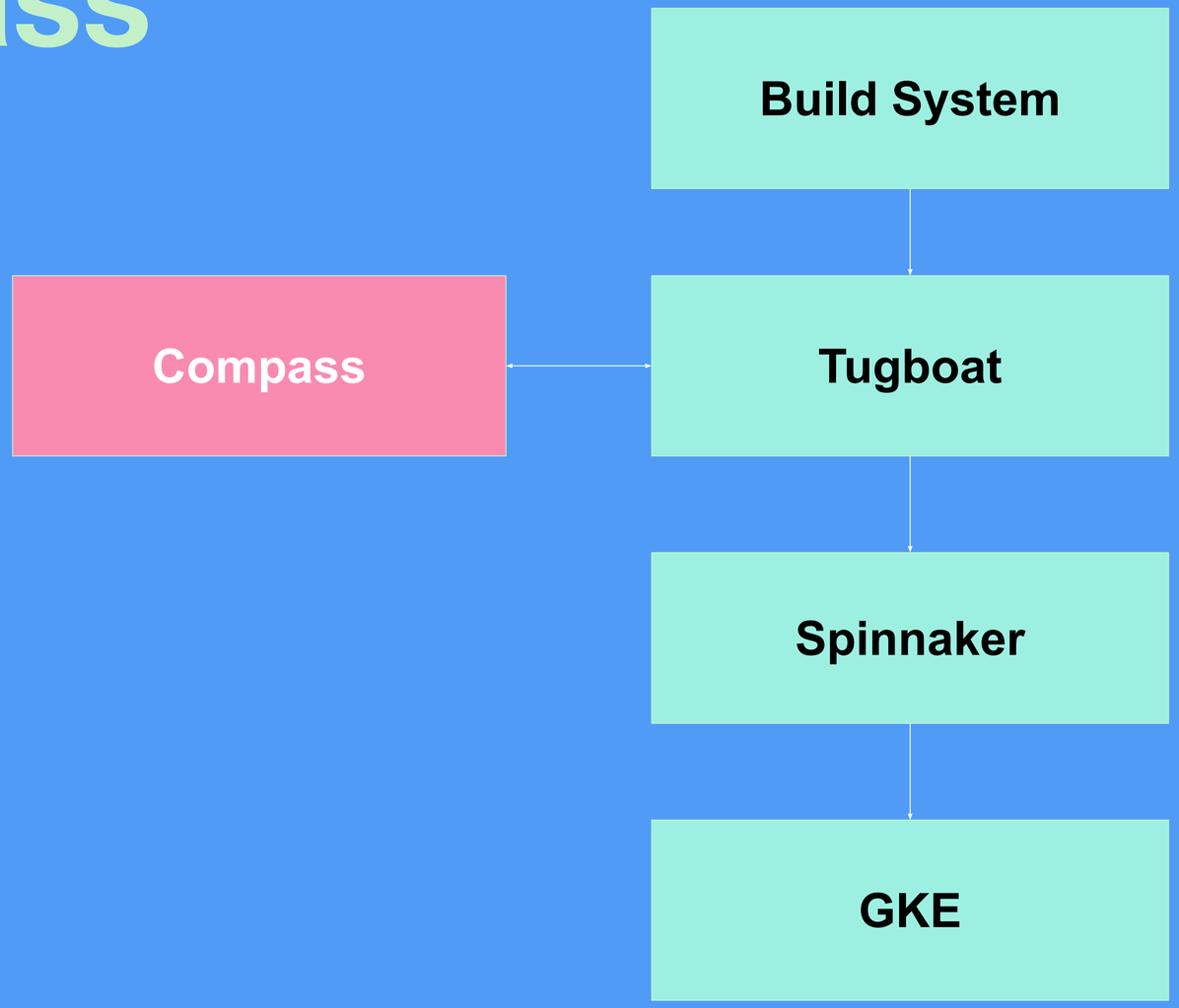
Tactics

Give developers visibility into their deployments at the levels they care about (reliability, observability)

Consider Knative for stateless applications

Plan Network topology early (maybe first!)

Enter Compass



Compass

```yaml
---
version: 1
components:
- componentId: anotherfoo
  tugboat:
    path: anotherfoo_service
    backends:
    - backend: helios
      jobConfig: anotherfoo_service/.helios/helios_jobs_are_great_config.
    - backend: kubernetes
      manifestsDir: anotherfoo_service/kubernetes-manifests-dir
      regions:
      - asia-east1
      - europe-west1
      canary:
        replicas: 1
      applicationGroup: my-group
      clustersPerRegion: 1
```

deployment.yaml

This is the file we use to specify service deployments

Compass

```yaml
---
version: 1
components:
- componentId: anotherfoo
  tugboat:
    path: anotherfoo_service
    backends:
    - backend: helios
      jobConfig: anotherfoo_service/.helios/helios_jobs_are_great_config.
    - backend: kubernetes
      manifestsDir: anotherfoo_service/kubernetes-manifests-dir
    regions:
    - asia-east1
    - europe-west1
    canary:
      replicas: 1
    applicationGroup: my-group
    clustersPerRegion: 1
```

This bit specifies whether you're deploying to Kubernetes and/or Helios

Compass

```
---
version: 1
components:
- componentId: anotherfoo
  tugboat:
    path: anotherfoo_service
    backends:
    - backend: helios
      jobConfig: anotherfoo_service/.helios/helios_jobs_are_great_config.
    - backend: kubernetes
      manifestsDir: anotherfoo_service/kubernetes-manifests-dir
      regions:
      - asia-east1
      - europe-west1
      canary:
        replicas: 1
      applicationGroup: my-group
      clustersPerRegion: 1
```

Here we allow developers to specify one or many regions for their service deployment

Compass

```yaml
---
version: 1
components:
- componentId: anotherfoo
  tugboat:
    path: anotherfoo_service
    backends:
    - backend: helios
      jobConfig: anotherfoo_service/.helios/helios_jobs_are_great_config.
    - backend: kubernetes
      manifestsDir: anotherfoo_service/kubernetes-manifests-dir
      regions:
      - asia-east1
      - europe-west1
      canary:
        replicas: 1
      applicationGroup: my-group
      clustersPerRegion: 1
```

Lastly, this is where developers define their per region replication

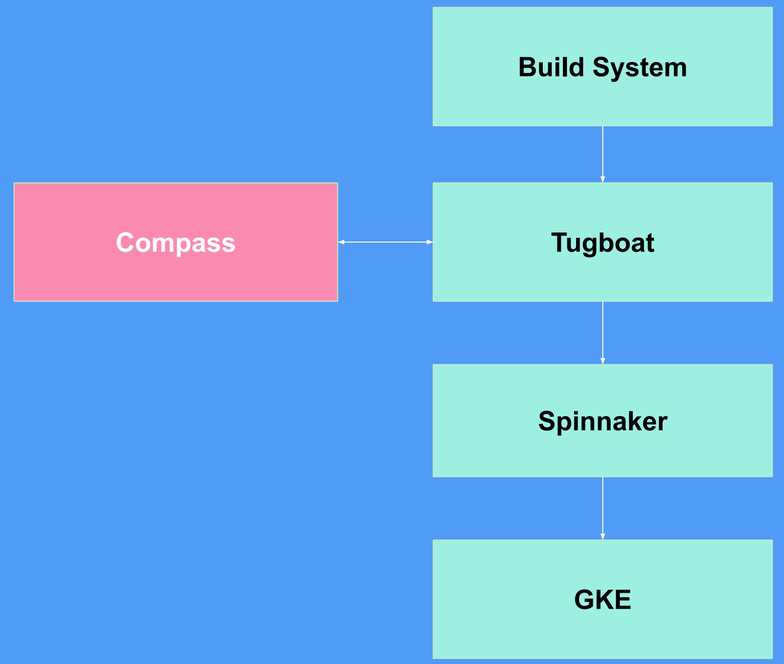
… but then there are many deployments … and things can get messy

Manage it!

Compass abstracts and manages:

- Which cluster a service is deployed to based on different algorithms

- If a group of services should be deployed to the same set of clusters together



Visualize Deployment

Currently running on kubernetes in production - Last updated at 1:20 PM

Legend

us-central1-b04t	production-europe-west1	asia-east1-k33l
pods: 2	pods: 2	pods: 2

We then visualize this for engineers in our developer portal and allow them to interface with their current versions and deployments *seamlessly*

Active Versions

Currently running in production - Last updated at 1:18 PM

Kubernetes

Search...

REGION	CLUSTER	INSTANCE	COMMIT	STATE	CONSOLE LINK	LOGS
asia-east1	asia-east1-k33l	compass-578456f	887e50ba	RUNNING		
asia-east1	asia-east1-k33l	compass-578456f	887e50ba	RUNNING		
europe-west	production-europe-west1	compass-578456f	887e50ba	RUNNING		
europe-west	production-europe-west1	compass-578456f	887e50ba	RUNNING		
us-central1	us-central1-b04t	compass-578456f	887e50ba	RUNNING		
us-central1	us-central1-b04t	compass-578456f	887e50ba	RUNNING		

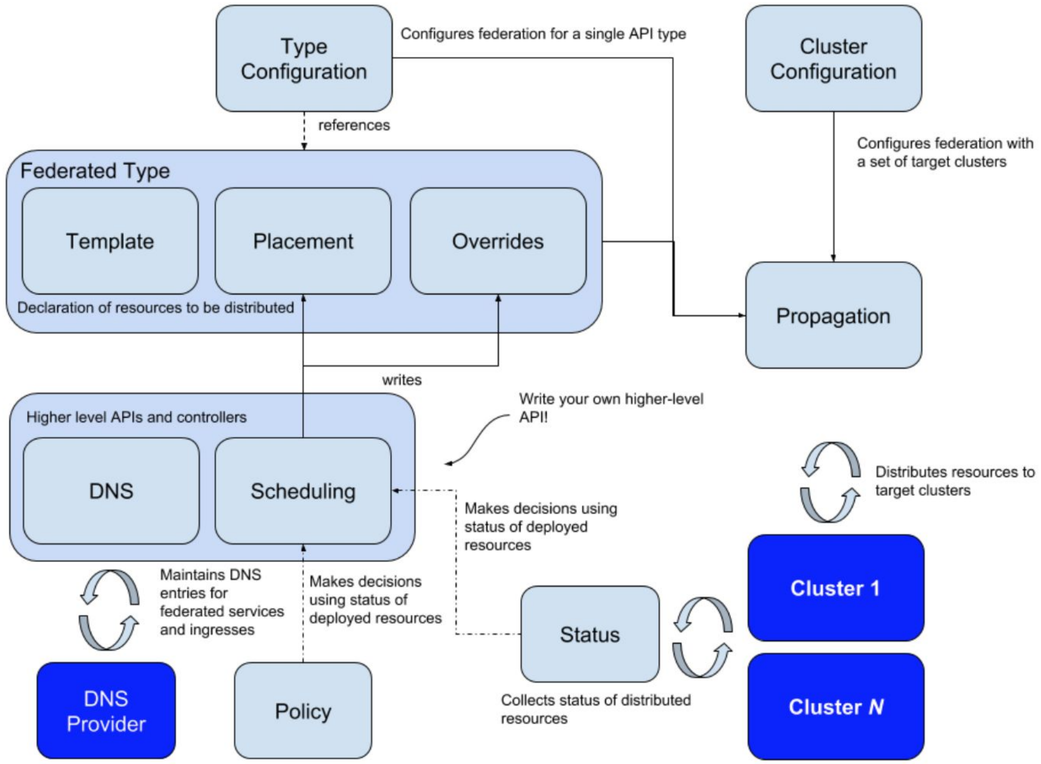
Alternative Solutions

Too Simple + Leads to Over-Provisioning

- Deploy every services to every cluster

Too Complex

- Kubernetes Cluster Federation
  - /kubernetes-sigs/kubefed



Concept diagram for Kubernetes Cluster Federation

Pitfalls

Start Simple

For Operators
If you don’t have this problem, don’t built a system like it!

Roll it out to a single cluster, then expand

Complexity is (mostly) Bad

For Developers
Make it easy to understand and do the simple thing by default

For Operators
Don’t build a complex abstraction or algorithm that you don’t need as you’ll just have to maintain it later… or worse reason about it in an incident

Takeaways

Things we hope you can take
and act on from this session

Automate Cluster Upgrades

Automate cluster upgrades for both minor upgrades and patch versions of Kubernetes and related extensions

Manage Multicluster Deployments

Create multi-tenant ephemeral clusters across regions and manage cross-cluster deployments

To infinity and beyond!