Navigating the Kubernetes Odyssey

Lessons from Early Adoption and Sustained Modernization

Raúl Benencia

Site Reliability Engineer

About ThousandEyes

Network intelligence platform

Early infrastructure

- First servers were scavenged from recycling bins
- Running from a garage in Mountain View
- Moved to a data center shortly after

- Growing at a faster pace than what we could scale
- Kubernetes comes into the picture. The year, 2015
- We didn't know if it would live up to the hype

ThousandEyes
part of Cisco

The journey begins

We start our research

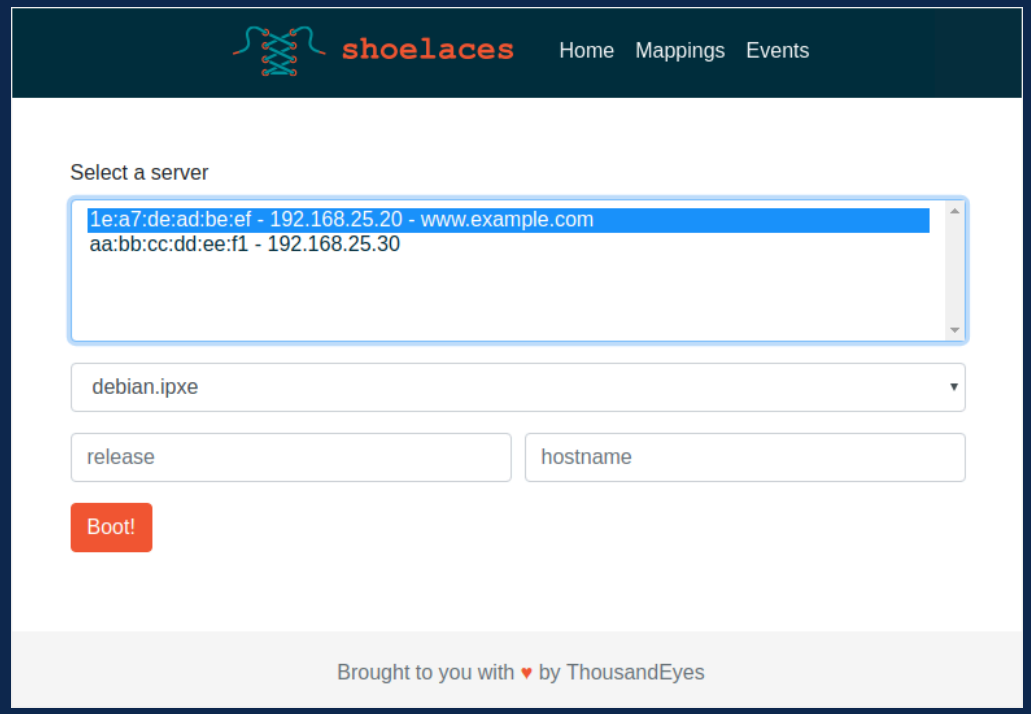
- The year, 2015
- First cluster:
  - Kubernetes version 1.1.2 (not even `Deployment` existed)
  - CoreOS
  - Ansible
- Named it... k8s1
- We hosted absolutely everything in our data center
- Each Kubernetes worker was a bare-metal server
- Control plane and etcd, VMs

From VMs to Containers

- Software footprint was small enough
- All teams collaborated in getting their workloads containerized.
- Containeraizing everything was a daunting but doable effort

Homegrown tool for automation

- Automation using Shoelaces
- Server bootstrapping automation tool, open sourced in 2018
- Uses DHCP attributes and supports hands-offs installations



`github.com/thousandeyes/shoelaces`

ThousandEyes
part of Cisco

The first challenge

Challenges with CoreOS

- We scaled. From 4 racks we went to 10 racks.
- Container optimized OS for a container orchestration system such as Kubernetes
- Not compatible with parts of our infrastructure
- Killer feature, rebooting on security upgrades, not in use
- The rest of our fleet was using Ubuntu, configured with Puppet
- CoreOS was not the right fit for us.

The first Kubernetes migration

- We decided to switch the OS, keeping the cluster
- We went for a modern infrastructure (at that time)
- We aligned with the rest of our servers and went back to Ubuntu
- Prepared Puppet modules, rounded a few sharp edges

Manual steps

- Remove CoreOS worker from load balancer, and from k8s cluster
- Use Shoelaces to re-bootstrap server with our new Ubuntu recipe
- Add Ubuntu worker to k8s cluster and to load balancer
- Repeat

Successful migration

whipped the tablecloth out from under our infra dinnerware and our developers barely felt it

ThousandEyes
part of Cisco

Jumping ships

Challenges

- Neglected version updates
- Stuck on Kubernetes 1.5 while 1.13 was out
- Internal tool used for Kubernetes manifest deployments
  - Triggered by Jenkins with changes in the `k8s` directory of a git repo
  - Needed robustness and flexibility
  - Drift detection was challenging

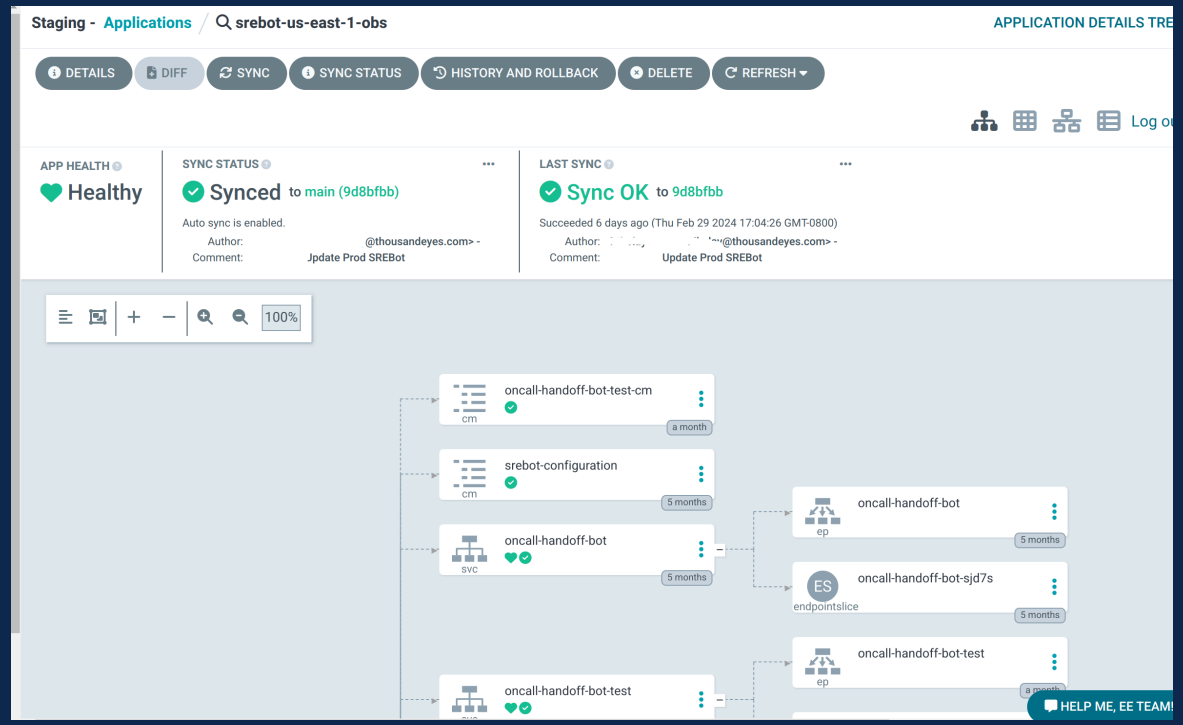
Trade-off: upgrading vs migrating

- Do we upgrade?
- Do we start from scratch in a new cluster?

A New Cluster Emerges

- Kubernetes 1.15.3 on Ubuntu
- Managed by Puppet
- We named it... k8s2

GitOps and ArgoCD



How did we migrate?

- Leveraged the full engineering team for migration
- Teams would migrate their own workloads, at will
- Account for dependencies
- Coordinated effort due to manageable workload and team size
- Big `Lift and shift` approach

ThousandEyes
part of Cisco

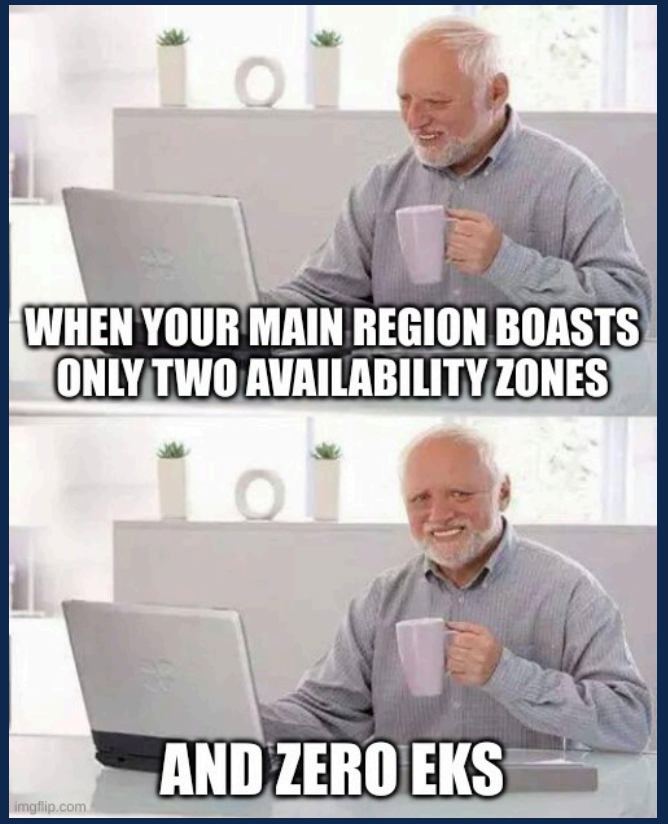
Sailing to the clouds

We keep scaling

- Our ten-rack data center was running small
- One region only, latency was high for oversea customers
- Single point of failure
- We needed to move. Our choice, AWS

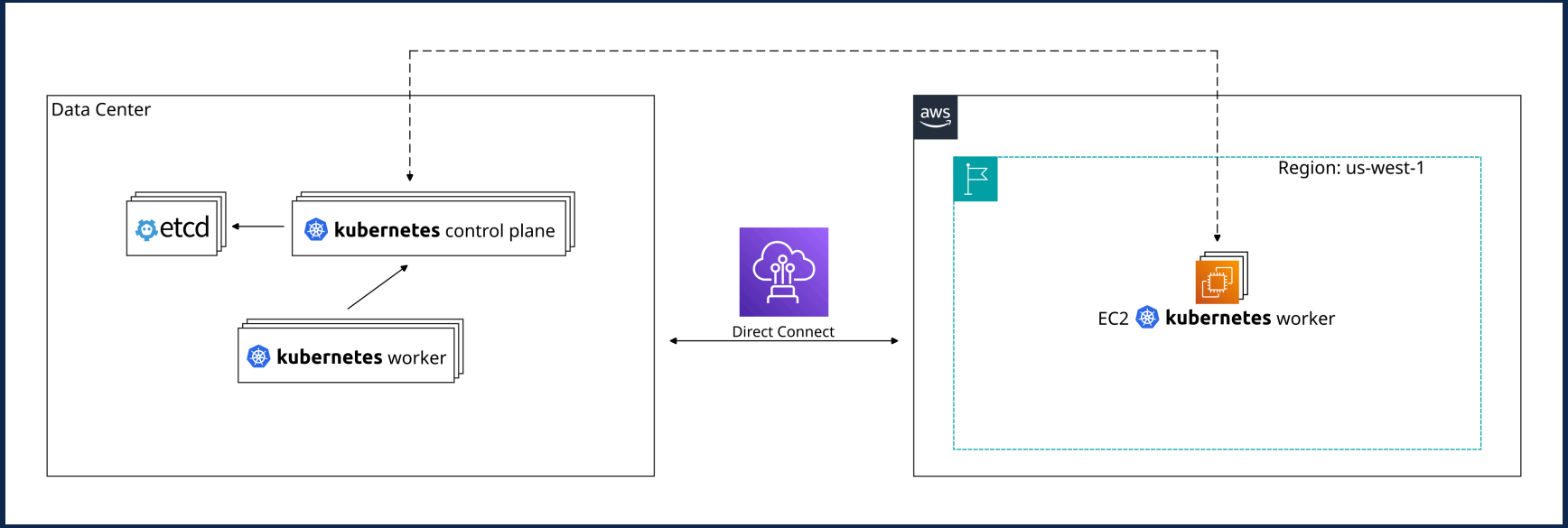
AWS Choices

- `us-west-1` as main region
- Direct Connect bridges to our data center
- Only 2 availability zones
- EKS was already a thing
- … but not in the region we chose



Extending k8s2

- We did not have a clear timeline for EKS support in `us-west-1`
- Decided to extended k8s2 by adding new EC2 workers (running in AWS)
- k8s2 EC2 workers in AWS communicated with data center control plane

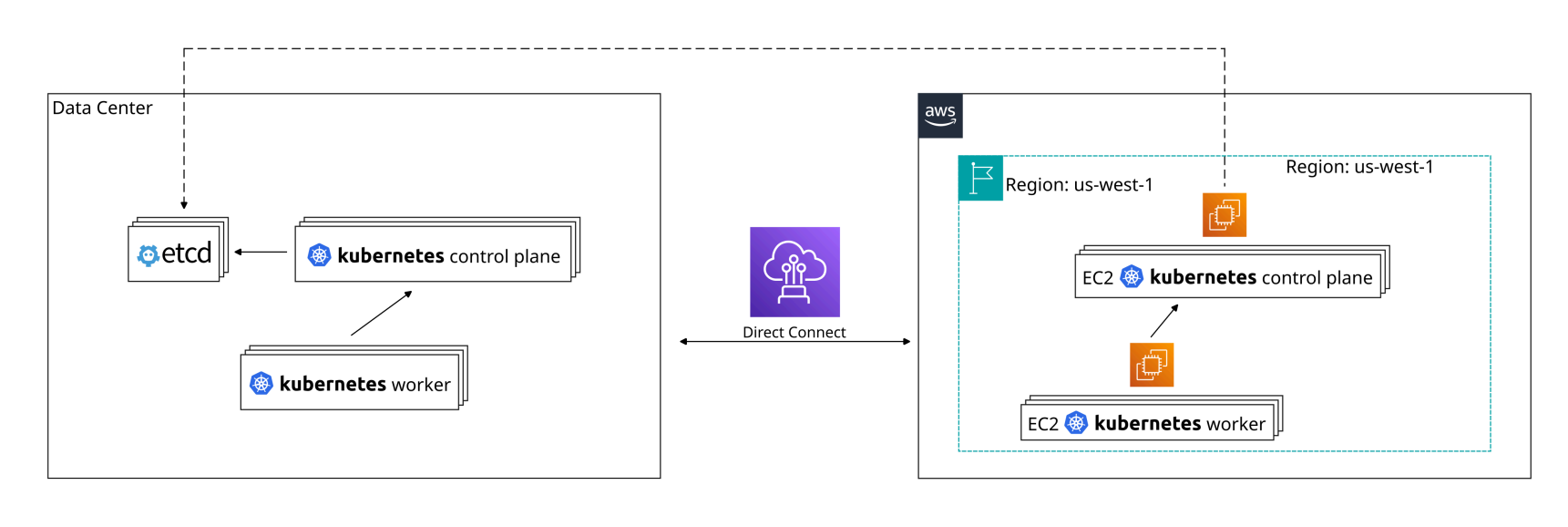


- Taints applied on AWS nodes to repel non-migrated workloads
- `taints`
  - Node repellents that prevent pods from landing unless they have the right `tolerations`
- `tolerations`
  - Pods tolerate node `taints` to be scheduled
  - Prevents pod assignment to unsuitable nodes
- Pods without corresponding tolerations remained in the existing cluster

```
$ kubectl describe no k8s2-1-a
Name:               k8s2-1-a
<...>
Taints:             site=sfo2:NoSchedule
```

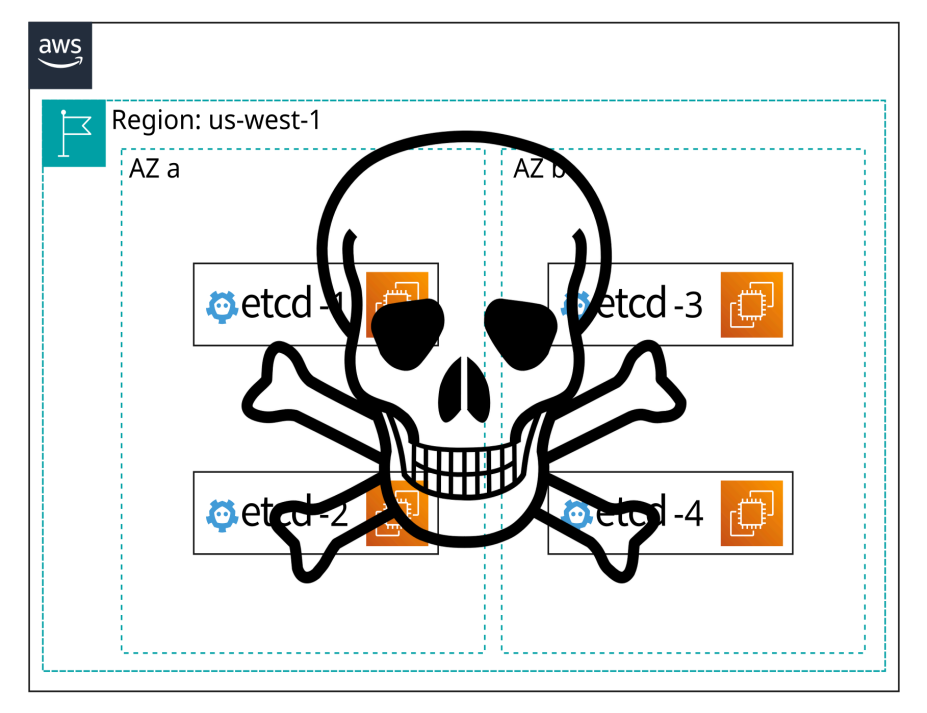
Kubernetes Control Plane Migration

- Added control plane nodes in AWS
- Maintained communication with the `etcd` cluster in the data center
- AWS workers connected to the new AWS control plane nodes



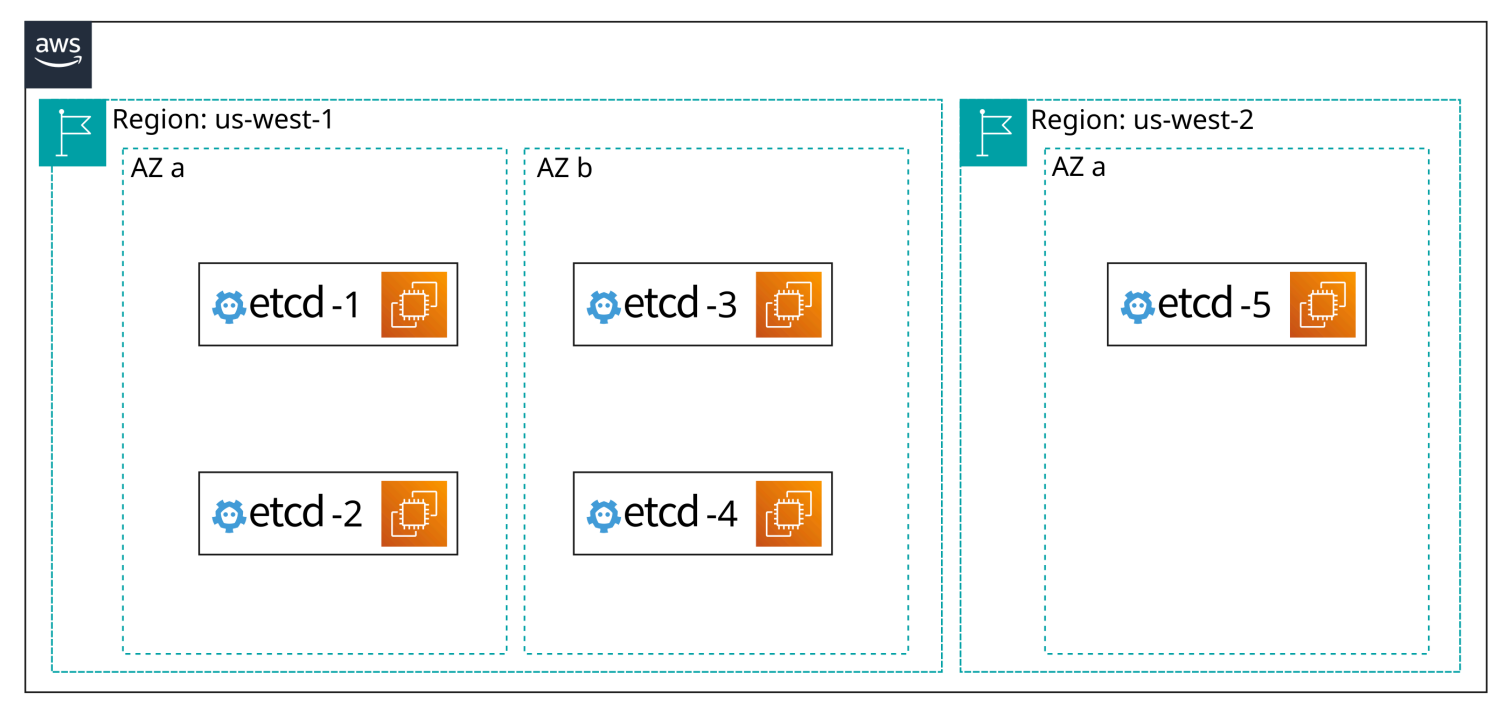
etcd Cluster Migration

- AWS control plane nodes talked with `etcd` in data center
- Migrating `etcd` from data center to AWS was required
- Challenge: `us-west-1` has only two availability zones
- Quorum and split-brain situation



Solution

- Add a node in `us-west-2`
- Two nodes in each `us-west-1` region
- k8s control plane pointed only to `us-west-1` nodes
- Synchronization with the fifth node happened in the background

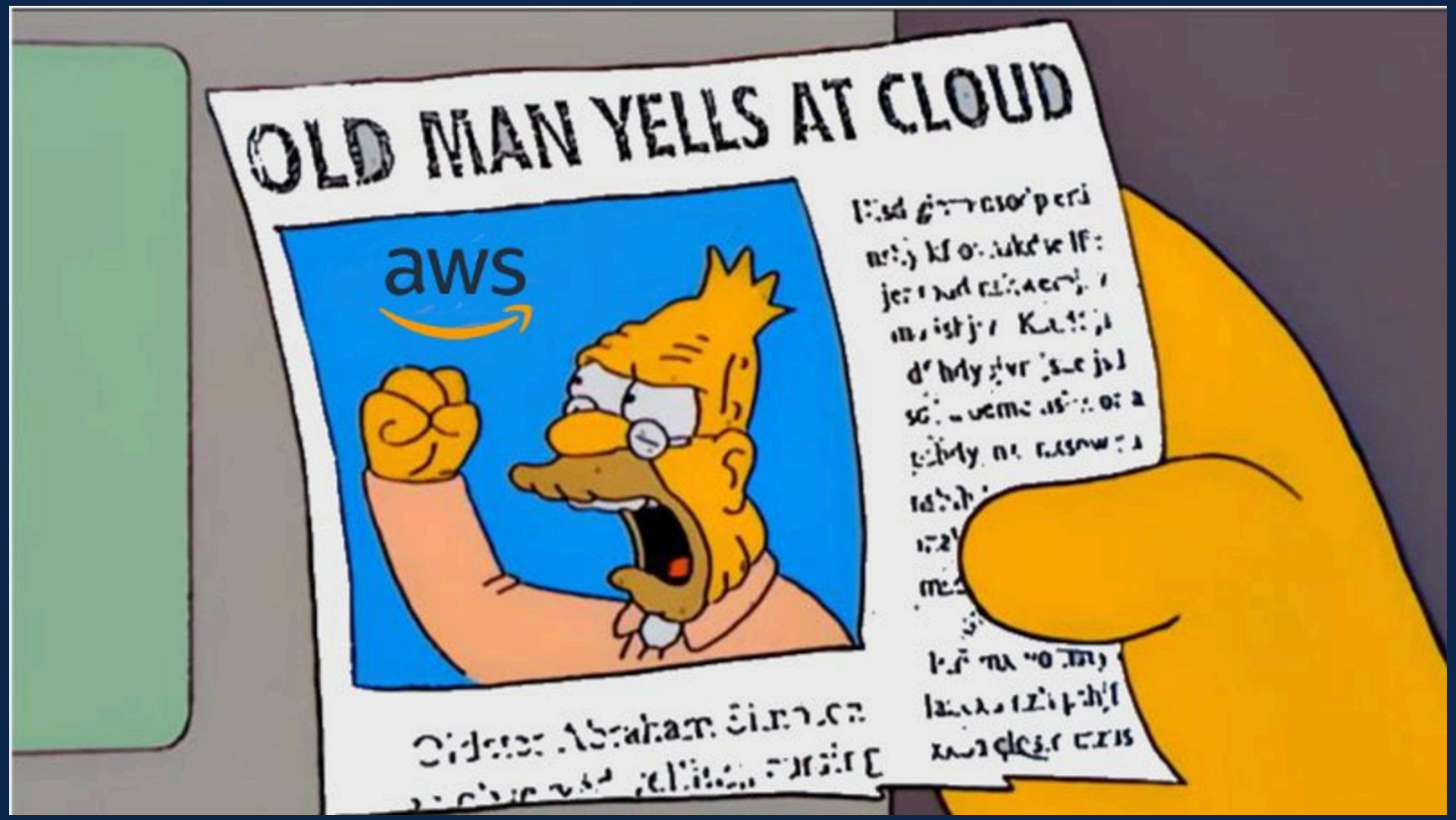


Workload Migrations to AWS

- Similar to the migration from `k8s1` to `k8s2`
- Piggybacking modernization and ASGs
- Engineering team called upon to migrate their workloads
- Teams had strict timelines for their own projects
- Not an approach that we could keep using

Mid-Migration Fun

EKS enabled in `us-west-1`



Trade-off: Complete migration or start anew?

- Sometimes it makes sense to do throw-away work.

ThousandEyes
part of Cisco

Testing new waters

Expanding Horizons: Into Europe

- Growing customer base in Europe
- Addressing latency issues with a new AWS region
- Chose `eu-central-1`
- This time, we had three availability zones and full EKS support

Terraforming EKS Clusters

- Developed Terraform code for EKS cluster bootstrap
- Aimed for creating whole setup with single pull request
- Challenges required code layering
  - Cluster bootstrap
  - Core services installation
  - Load balancer setup

Launching `eks1`

- Established the new EKS cluster.
- We named it… **`eks1`**
- Engineers easily deployed services to EU cluster
- Began serving European customers with reduced latency

Demand for clusters

- Surge in requests for new EKS clusters post-initial launch
- Disaster recovery, team-specific, and tool-specific needs
- Iterative improvement of our cluster bootstrapping process

However...

- Despite uniformity, one cluster remained an exception

ThousandEyes
part of Cisco

The sinking ship

Trade-off: Maintenance vs. Innovation

- Main SaaS platform still hosted on legacy cluster `k8s2`
- Small team focusing on developing EKS modules
- `k8s2` maintenance became secondary, leading to version lag
- EKS clusters maintained with consistent update cadence
- Strategic shift to plan for migration to EKS

Inconsistency Rains

- Deprecated APIs and new features caused environment drift
- Resorted to `kustomize` patches for temporary fixes

Example of using `kustomize` to patch Ingress API version

```yaml
patchesJson6902:
  - target:
      group: networking.k8s.io
      version: v1
      kind: 'Ingress'
      name: '.*'
    patch: |-
      - op: replace
        path: "/apiVersion"
        value: networking.k8s.io/v1beta1
```

The Sinking Flagship

- `k8s2` facing a hard limit of 255 nodes, nearing capacity
- We could have fixed this, but we wanted to avoid throw-away work.
- Transition to EKS for uniformity and scalability

ThousandEyes
part of Cisco

The gangway

A New Migration Challenge

- Did we need to actively involve all teams in a new cluster-to-cluster migration from `k8s2` to `eks1`?
- We've scaled to hundreds of services and dozens of teams
- We needed a different approach to avoid a migration nightmare

Integrating a Service Mesh

- We always lacked the right "excuse" to implement a service mesh
- Seized the migration to implement it
- Services to communicate with each other regardless of cluster
- Istio chosen for its maturity and community support
- Installed on `k8s2`
- Bootstrapped new EKS cluster `eks1` in the same AWS region

Bridging Clusters with Istio

- Set up east-west gateway to connect `k8s2` and `eks1`
- Established a "gangway" for smooth service transition

```yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: istio-eastwestgateway
  namespace: istio-system
spec:
  selector:
    istio: eastwestgateway
  servers:
  - hosts:
    - '*.local'
    port:
      name: tls
      number: 15443
      protocol: TLS
```

Enabling Istio Across Workloads

- Istio installed and gateway established in clusters
- Services integrated into the mesh namespace by namespace
- Used scripts to enable Istio in nss and restart workloads

```
if [ "$OP" == "enable" ]; then
  kubectl label --overwrite ns "$NS" istio.io/rev=$ISTIO_REVISION
elif  [ "$OP" == "disable" ]; then
  kubectl label ns "$NS" istio.io/rev-
fi
...
for i in $items; do
  sleep "$TIMER" &
  if [ $objtype == 'rollout' ]; then
    kubectl argo rollouts restart "$i" -n "$NS"
  else
    kubectl rollout restart "$objtype" "$i" -n "$NS"
  fi
  wait
done
```

Traffic Control with Istio

- Parallel workloads started in `eks1`
- Subsets in `DestinationRules` for workloads in different networks
- Controlled traffic distribution with `VirtualServices`

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: webapps-ing-ctrl.webapps.svc.cluster.local
spec:
  host: webapps-ing-ctrl.webapps.svc.cluster.local
  subsets:
    - labels:
        topology.istio.io/network: k8s2.prd.sfo2
      name: k8s2
    - labels:
        topology.istio.io/network: eks1.prd.sfo2
      name: eks1
---
apiVersion: networking.istio.io/v1beta1
```

Streamlined Migration Automation

- Automated manifest generation for workload migration
- Ensured smooth, programmatic transition of services

```
$ ./bin/generate-migration-manifests \
    --src-env-dir environments/staging/us-west-1/k8s2/ \
    --dst-env-dir environments/staging/us-west-1/eks1/ \
    --kube-ctx k8s2.stg.sfo2 \
    --ns agent --svc agent-service
[+] Generating yaml for agent-service.agent in environments/staging/us-west-1/k8s2/agent/ag
[+] Generating yaml for agent-service.agent in environments/staging/us-west-1/eks1/agent/ag
```

Collaborative yet Independent Migration

- Executed migration over several weeks, one namespace at a time
- Kept parity in staging and production migrations
- Achieved migration of hundreds of workloads across dozen of teams
- Limited engineering-wide involvement through efficient automation
- Set the stage for final decommissioning of `k8s2`

Migration successful

`k8s2` served us well until the end

ThousandEyes
part of Cisco

Conclusion

Our Kubernetes journey

Presented a journey of growth and expansion in Kubernetes usage

- `k8s1`: Early baremetal and CoreOS-based
- `k8s2`: Modernized baremetal with Ubuntu, configured with Puppet
- `k8s2.aws`: Modernized `k8s2` in AWS with ASGs
- `eks1`: Terraformed EKS
- Myriad of EKS clusters

- EKS Modernized with:
  - `keda`, paramount for event-based scaling
  - `karpenter`: bin-packing pods into nodes
  - `gatekeeper`: OPA policies
- Our hopeful future:
  - Fine-tuned service mesh
  - One-click cluster provisioning

The Balancing Act

- Challenges faced by a small team managing large-scale infrastructure
- Navigate operations with sustainable practices

Managed Clusters: Not a panacea

- Managed clusters reduce overhead but are not a cure-all
- Need for vigilance in keeping manifests up-to-date
- In large environments, manifests automation is a **must**

Steady Progress

- Adapt strategies to the organization's maturity and resources
- Make decisions based on size and workload diversity
- Embrace continuous learning and incremental improvements
- Learn, improve, advance. Cluster by cluster

Thanks!