Kubernetes the Very Hard Way

Laurent Bernaille
Staff Engineer, Infrastructure
@lbernail
Datadog

Over 350 integrations
Over 1,200 employees
Over 8,000 customers
Runs on millions of hosts
Trillions of data points per day

10,000s hosts in our infra
10s of k8s clusters with 50-2500 nodes
Multi-cloud
Very fast growth
# Why Kubernetes?

<table>
<thead>
<tr>
<th>Dogfooding</th>
<th>Immutable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improve k8s integrations</td>
<td>Move from Chef</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multi Cloud</th>
<th>Community</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common API</td>
<td>Large and Dynamic</td>
</tr>
</tbody>
</table>
The very hard way?

Bootstrap Kubernetes the hard way on Google Cloud Platform. No scripts.
It was much harder
This talk is about the fine prints

“Of course, you will need a HA master setup”

“Oh, and yes, you will have to manage your certificates”

“By the way, networking is slightly more complicated, look into CNI / ingress controllers”
What happens after “Kube 101”

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
   a. Kubernetes and Certificates
   b. Exceptions?
   c. Impact of Certificate Rotation
3. Efficient networking
   a. Giving pod IPs and routing them
   b. Accessing services: Client-side load-balancing:
   c. Ingresses: Getting data in the cluster
What happens after “Kube 101”

1. Resilient and Scalable Control Plane
   a. Kubernetes and Certificates
   b. Exceptions?
   c. Impact of Certificate Rotation

2. Securing the Control Plane
   a. Kubernetes and Certificates
   b. Exceptions?
   c. Impact of Certificate Rotation

3. Efficient networking
   a. Giving pod IPs and routing them
   b. Accessing services: Client-side load-balancing:
   c. Ingresses: Getting data in the cluster
Resilient and Scalable Control Plane
Kube 101 Control Plane

- etcd
- apiserver
- scheduler
- controllers
- kubelet
- kubectl
Making it resilient
Separate etcd nodes
Single active Controller/scheduler
Split scheduler/controllers

Diagram showing the relationship between etcd, apiserver, LoadBalancer, scheduler, kubelet, kubectl, and controllers.
What happens after “Kube 101”

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
   a. Kubernetes and Certificates
   b. Exceptions?
      c. Impact of Certificate Rotation
3. Efficient networking
   a. Giving pod IPs and routing them
   b. Accessing services: Client-side load-balancing:
      c. Ingresses: Getting data in the cluster
Kubernetes and Certificates
From “the hard way”

cat > ca-config.json
{
    "signing": {
        "default": {
            "expiry": "8760h"
        },
        "profiles": {
            "kubernetes": {
                "usages": ["signing", "key encipherment", "server auth", "client auth"],
                "expiry": "8760h"
            }
        }
    }
}
"Our cluster broke after ~1y"

cat > ca-config.json
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": ["signing", "key encipherment", "server auth", "client auth"],
        "expiry": "8760h"
      }
    }
  }
}
Certificates in Kubernetes

- Kubernetes uses certificates everywhere
- Very common source of incidents
- Our Strategy: Rotate all certificates daily
Certificate management

- etcd
- apiserver
- etcd PKI
- Peer/Server cert
- Etcd Client cert

Vault

Ibernail
Certificate management

- etcd
  - Peer/Server cert
  - Etcd Client cert
- etcd PKI
- Vault
- apiserver
  - etcd PKI
- kube PKI
  - Apiserver/kubelet client cert
  - Controller client cert
  - Scheduler client cert
  - Kubelet client/server cert
- controllers
- scheduler
- kubelet
Certificate management

- etcd
  - Peer/Server cert
  - etcd PKI
  - etcd PKI

- apiserver
  - Etcd Client cert
  - Apiserver/kubelet client cert
  - kube PKI
  - kube PKI

- controllers
  - SA public key
  - SA private key
  - kubectl
  - Controller client cert
  - Scheduler client cert

- scheduler
  - SA token
  - Scheduler client cert

- kubelet
  - In-cluster app
  - Kubelet client/server cert

- In-cluster app
  - kubelet
  - kubectl

Vault
Certificate management

- etcd
- apiserver
- controllers
- scheduler
- kubelet
- In-cluster app
- apiservice webhook...
- etcd PKI
- apiservice PKI
- kube PKI
- kube kv

- Peer/Server cert
- Etcd Client cert
- Apiservice cert (proxy/webhooks)
- Apiserver/kubelet client cert
- SA public key
- SA private key
- Controller client cert
- Scheduler client cert
- Kubelet client/server cert
Certificate management

- OIDC provider
- etcd
- apiserver
- controllers
- scheduler
- kubelet
- In-cluster app
- apiservice webhooks...
- etcd PKI
- apiservice PKI
- kube PKI
- kube kv
- Vault
- OIDC auth
- SA token
- SA public key
- SA private key
- Peer/Server cert
- Etd Client cert
- Apiservice cert (proxy/webhooks)
- Apiserver/kubelet client cert
- Controller client cert
- Scheduler client cert
- Kubelet client/server cert
Exception ?
Incident...
Kubelet: TLS Bootstrap

1- Create Bootstrap token
2- Add Bootstrap token to vault
3- Get signing key
Kubelet: TLS Bootstrap

1- Get Bootstrap token

2- Authenticate with token
4- Create CSR

3- Verify Token and map groups
5- Verify RBAC for CSR creator
6- Sign certificate

7- Download certificate
8- Authenticate with cert
9- Register node

apiserver
controllers
kubelet
kube PKI
kube kv
Vault

Ibernail
Kubelet certificate issue

1. One day, some Kubelets were failing to start or took 10s of minutes
2. Nothing in logs
3. Everything looked good but they could not get a cert
4. Turns out we had a lot of CSRs in flight
5. Signing controller was having a hard time evaluating them all

CSR resources in the cluster
Lower is better!
Why?

Initial creation
1. Authenticate with bootstrap token, mapped to group “system:bootstrappers”
2. Create CSR
3. “system:bootstrappers” has role “system:certificates.k8s.io:certificatesigningrequests:nodeclient”

Renewal
1. Authenticate with current node certificate, mapped to group “system:nodes”
2. Create CSR
3. Not allowed for auto-sign

Also needed for “system:nodes”
Exception 2?
Incident 2...
Temporary solution

Create webhook with self-signed cert as CA

Get cert and key

Add self-signed cert + key to Vault

One day, after ~1 year

- Creation of resources started failing (luckily only a Custom Resource)
- Cert had expired...
Take-away

- Rotate server/client certificates
- Not easy

But, “If it’s hard, do it often”

> no expiration issues anymore
Impact of Certificate rotation
Apiserver restarts

We have multiple apiservers
We restart each daily

Significant etcd network impact
(caches are repopulated)

Significant impact on etcd performances
Apiserver restarts, continued

- Apiserver restarts
- Clients reconnect and refresh their cache

> Memory spike for impacted apps

No real mitigation today
Unbalanced apiserver traffic

Number of connections / traffic very unbalanced
Because connections are very long-lived

More clients => Bigger impact clusterwide

15MB/s
2.5MB/s

2300 connections
300 connections
Take-away

Restarting components is not transparent
- Not limited to apiservers, some issues with the Kubelet too

It would be great if
- Components could transparently reload certs (server & client)
- Clients could wait 0-Xs to reconnect to avoid thundering herd
- Reconnections did not trigger memory spikes
- Connections were rebalanced (kill them after a while?)
What happens after “Kube 101”

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
   a. Kubernetes and Certificates
   b. Exceptions?
   c. Impact of Certificate Rotation
3. Efficient networking
   a. Giving pod IPs and routing them
   b. Accessing services: Client-side load-balancing:
   c. Ingresses: Getting data in the cluster
Efficient networking
Network challenges

**Throughput**
Trillions of data points daily

**Latency**
End-to-end pipeline

**Scale**
1000-2000 nodes clusters

**Topology**
Multiple clusters
Access from standard VMs
Giving pods IPs & Routing them
From “the Hard Way”

Routes

Create network routes for each worker instance:

```bash
for i in 0 1 2; do
gcloud compute routes create kubernetes-route-10-200-$i-0-24 \    --network kubernetes-the-hard-way \    --next-hop-address 10.240.0.2$i \    --destination-range 10.200.$i.0/24
done
```

- node IP
- Pod CIDR for this node
Small cluster? Static routes

Node 1
IP: 192.168.0.1
Pod CIDR: 10.0.1.0/24

Node 2
IP: 192.168.0.2
Pod CIDR: 10.0.2.0/24

Routes (local or cloud provider)
10.0.1.0/24 => 192.168.0.1
10.0.2.0/24 => 192.168.0.2
Mid-size cluster? Overlay

Node 1
IP: 192.168.0.1
Pod CIDR: 10.0.1.0/24

Node 2
IP: 192.168.0.2
Pod CIDR: 10.0.2.0/24

VXLAN

Tunnel traffic between hosts
Examples: Calico, Flannel
Large cluster with a lot of traffic? Native pod routing

Performance

Datapath: no Overlay
Control plane: simpler

Addressing

Pod IPs are accessible from
- Other clusters
- VMs
In practice

<table>
<thead>
<tr>
<th>On premise</th>
<th>GCP</th>
<th>AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGP</td>
<td>IP aliases</td>
<td>Additional IPs on ENIs</td>
</tr>
<tr>
<td>Calico</td>
<td></td>
<td>AWS EKS CNI plugin</td>
</tr>
<tr>
<td>Kube-router</td>
<td></td>
<td>Lyft CNI plugin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cilium ENI IPAM</td>
</tr>
</tbody>
</table>
A bit more complex on AWS

AWS

Attach ENI
Allocate IPs

agent

kubelet

containerd

cni

Pod 1

Pod 2

eth0

ip 1

eth1

Create veth

Routing rule
“From IP1, use eth1”

Routing

Routing rule

eth0

ip 1

ip 2

ip 3
Take-away

- Native pod routing has worked very well at scale
- A bit more complex to debug
- Much more efficient datapath
- Topic is still dynamic (Cilium introduced ENI recently)
- Great relationship with Lyft / Cilium
Kube-proxy default: iptables

# Mid size cluster
iptables -S -t nat | wc -l
48688

Kube-proxy facing locking timeout in large clusters during load test with services enabled #48107
Alternative: IPVS

Service ClusterIP:Port S
Backed by pod:port X, Y, Z

Virtual Server S
Realservers
- X
- Y
- Z

kube-proxy

Watches endpoints
Updates IPVS

apiserver

Pod X

Pod Y

Pod Z
New connection

App establishes connection to S
IPVS associates Realserver X
Pod X deleted

Apiserver removes X from S endpoints
Kube-proxy removes X from realservers
Kernel drops traffic (no realserver)
Pod X deleted

Pod X sends FIN on exit
Conntrack entry deleted
Connection from A terminates
What if X doesn’t send FIN?

Traffic blackholed until App detects issue
> tcp_retries2 (default 15)
> ~15mn
Mitigation

Virtual Server S
- Y
- Z

kube-proxy

apiserver

App

IPVS conntrack
A => S >> A => X

net/ipv4/vs/expire_nodest_conn
Delete conntrack entry on next packet
Forcefully terminate (RST)
Limit?

- No graceful termination
- As soon as a pod is terminating, connections are destroyed
- Addressing this took time
Graceful termination

Apiserver removes X from S endpoints
Kube-proxy sets Weight to 0
No new connection
Garbage collection

Pod exits and sends FIN
Kube-proxy removes realserver when it has no active connection
What if X doesn’t send FIN?

Conntrack entries expires after 900s
If A sends traffic, it never expires
Traffic blackholed until App detects issue
~15mn
> Mitigation: lower tcp_retries2
Take-away

- IPVS has been great for us
- IPVS is in a good state now
- Several improvements in the works
- But harder than we expected
- I ended up reviewer/approver for kube-proxy/IPVS
Ingresses
Ingress: cross-clusters, VM to clusters
Kubernetes default: LB service
Inefficient Datapath & cross-application impacts

- **Master**
  - service-controller

- **Web traffic**
  - Load-Balancer

- **Healthchecker**
  - data path
  - health checks
  - configuration (from watching ingresses on apiservers)

- **Load-Balancer**
  - NP kube-proxy
  - web-1
  - web-2
  - web-3
  - kafka
ExternalTrafficPolicy: Local?

Master
  service-controller

Web traffic
  Load-Balancer

Healthchecker

NP kube-proxy
web-1
web-2
web-3
NP kube-proxy
NP kube-proxy
kafka

---
data path
dashed health checks
configuration (from watching ingresses on apiservers)
L7-proxy ingress controller

- Configuration from watching ingresses/endpoints on apiservers (ingress-controller)
- From watching LoadBalancer services (service-controller)

Data path:
- Create l7proxy deployments
- Update backends using service endpoints

Health checks:
- From external client
- From LoadBalancer

Configuration:
- From watching ingresses/endpoints on apiservers (ingress-controller)
- From watching LoadBalancer services (service-controller)
Challenges

Limits

All nodes as backends (1000+)
Inefficient datapath
Cross-application impacts

Alternatives?

ExternalTrafficPolicy: Local?
  > Number of nodes remains the same
  > Issues with some CNI plugins
K8s ingress
  > Still load-balancer based
  > Need to scale ingress pods
  > Still inefficient datapath
Our target: native routing

-alb-ingress-controller

Healthchecker

External Client

ALB

pod

pod

pod

data path

health checks

configuration (from watching ingresses/endpoints on apiservers)
Remaining challenges

Limited to HTTP ingresses

- No support for TCP/UDP
- Ingress v2 should address this

Registration delay

- Slow registration with LB
- Pod rolling-updates much faster

Mitigations
- MinReadySeconds
- Pod ReadinessGates
Workaround

TCP / Registration delay not manageable

> Dedicated gateways

- External Client
- Load-Balancer
- Healthchecker
- l7proxy
- Dedicated nodes
  - Pods in host network

Not managed by k8s
Take-away

- Ingress solutions are not great at scale yet
- May require workarounds
- Definitely a very important topic for us
- The community is working on v2 Ingresses
Conclusion
A lot of other topics

- DNS (it’s always DNS!)
- Challenges with Stateful applications
- How to DDOS `<insert ~anything> with Daemonsets`
- Node Lifecycle
- Cluster Lifecycle
- Deploying applications
- ...
You want more horror stories?

“Kubernetes the very hard way at Datadog”
https://www.youtube.com/watch?v=2dsCwp_j0yQ

“10 ways to shoot yourself in the foot with Kubernetes”
https://www.youtube.com/watch?v=QKI-JRs2RIE

“Kubernetes Failure Stories”
https://k8s.af
Key lessons

Self-managed Kubernetes is hard
> If you can, use a managed service

Networking is not easy (especially at scale)

The main challenge is not technical
> Build a team
> Transforming practices and training users is very important
Thank you

We’re hiring!
https://www.datadoghq.com/careers/

laurent@datadoghq.com

@lbernnail