OSSO

kubernetes
in production

Meetup DockerGrunn 10 April 2017

Herman Bos <hbos@osso.nl>
Alex Boonstra <aboonstra@osso.nl>
Jelle Prins <jprins@osso.nl>

Plan

- Introducing OSSO
- Overview infrastructure setup
- Current k8s deployment
- Adding kubernetes to the mix
- Some important k8s concepts
- Demo cluster for tonight
- Protips
- Q&A

Introducing OSSO

- specializes in open source {hosting,network,cloud} infrastructure
- {dev,netdev,sys,sec}ops

Our customers:

- SaaS/cloud services, hosting providers and tech-startups

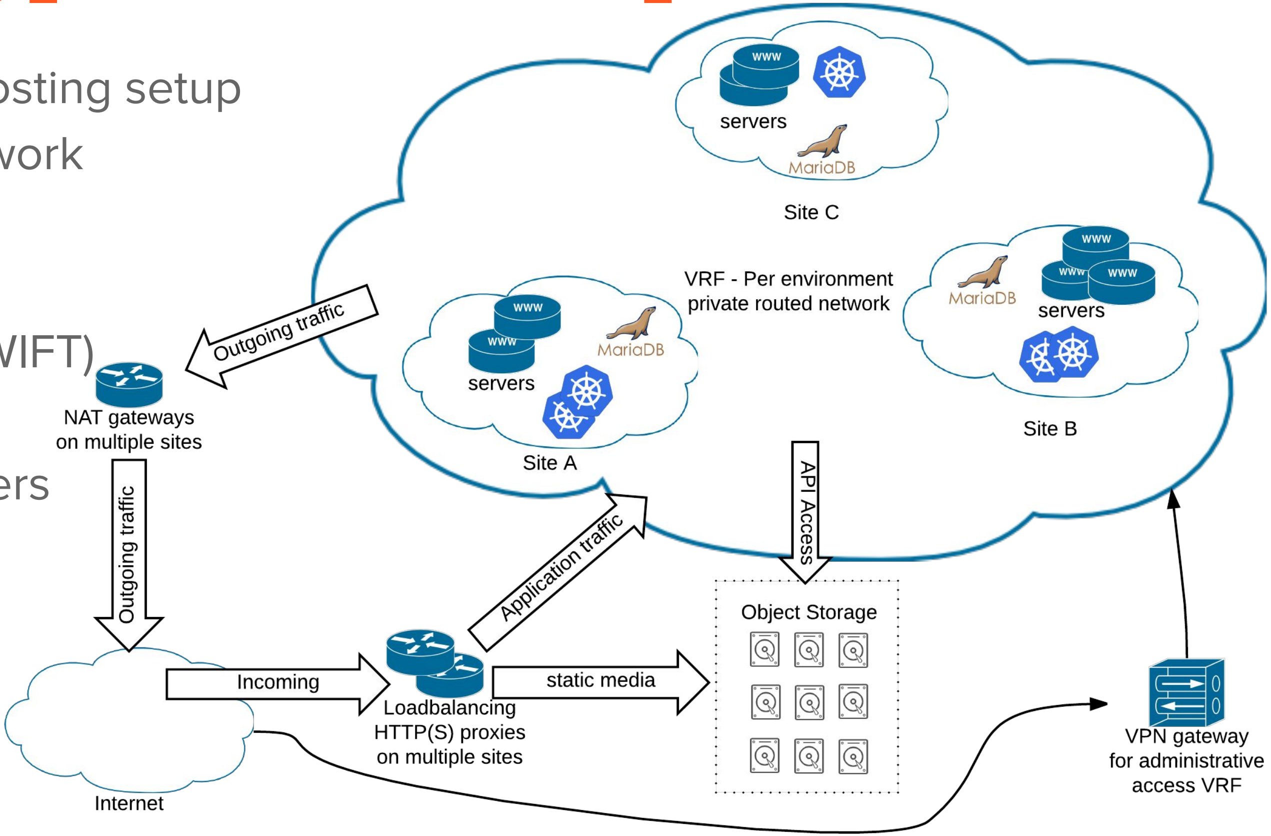
We provide:

- 24/7 operations and support.
- Expertise, infrastructure development and so on.
- Multi datacenter high availability hosting infrastructure.

Overview typical osso setup

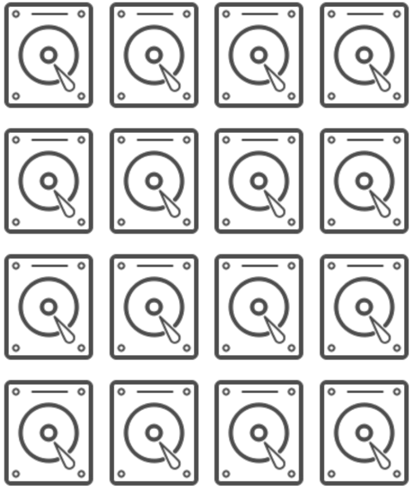
- Multi datacenter hosting setup
- Private routed network
- Loadbalancer
- Nat gateway
- Object Storage (SWIFT)
- VPN
- {micro,virtual} servers

Mostly offered as "IaaS"



Storage

- Object Storage for files (SWIFT, S3)
  - Cheap, no maintenance, high available fix.
- Other data in their respective services
  - MySQL, Redis, MongoDB, RabbitMQ, etc
  - Most common stuff can be deployed in a High availability configuration



PATENT PENDING - 1000 BASE T - TESTED
6
4
B1
B1
0907

Our current deployment

- Current k8s deployments
    - 4 production clusters
    - 3-5 kubernetes nodes each currently

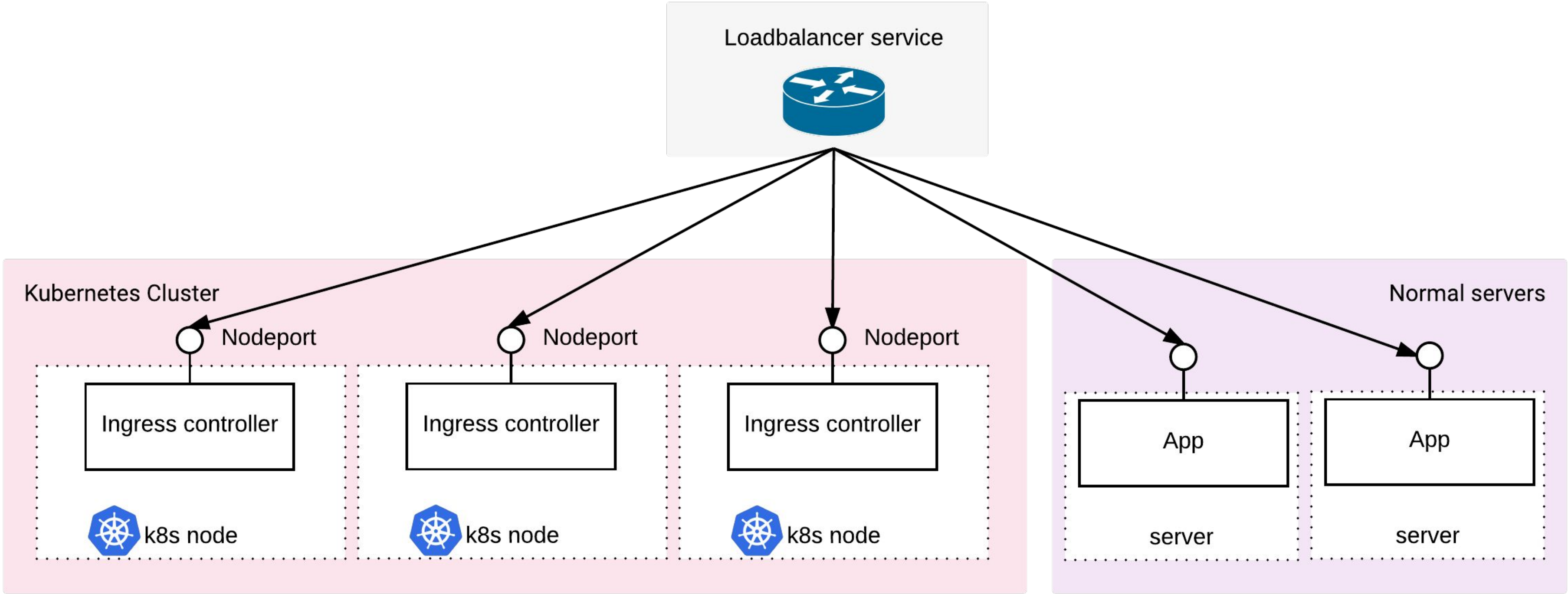
- We deploy k8s with saltstack on ubuntu 16.04lts

Deploying k8s itself

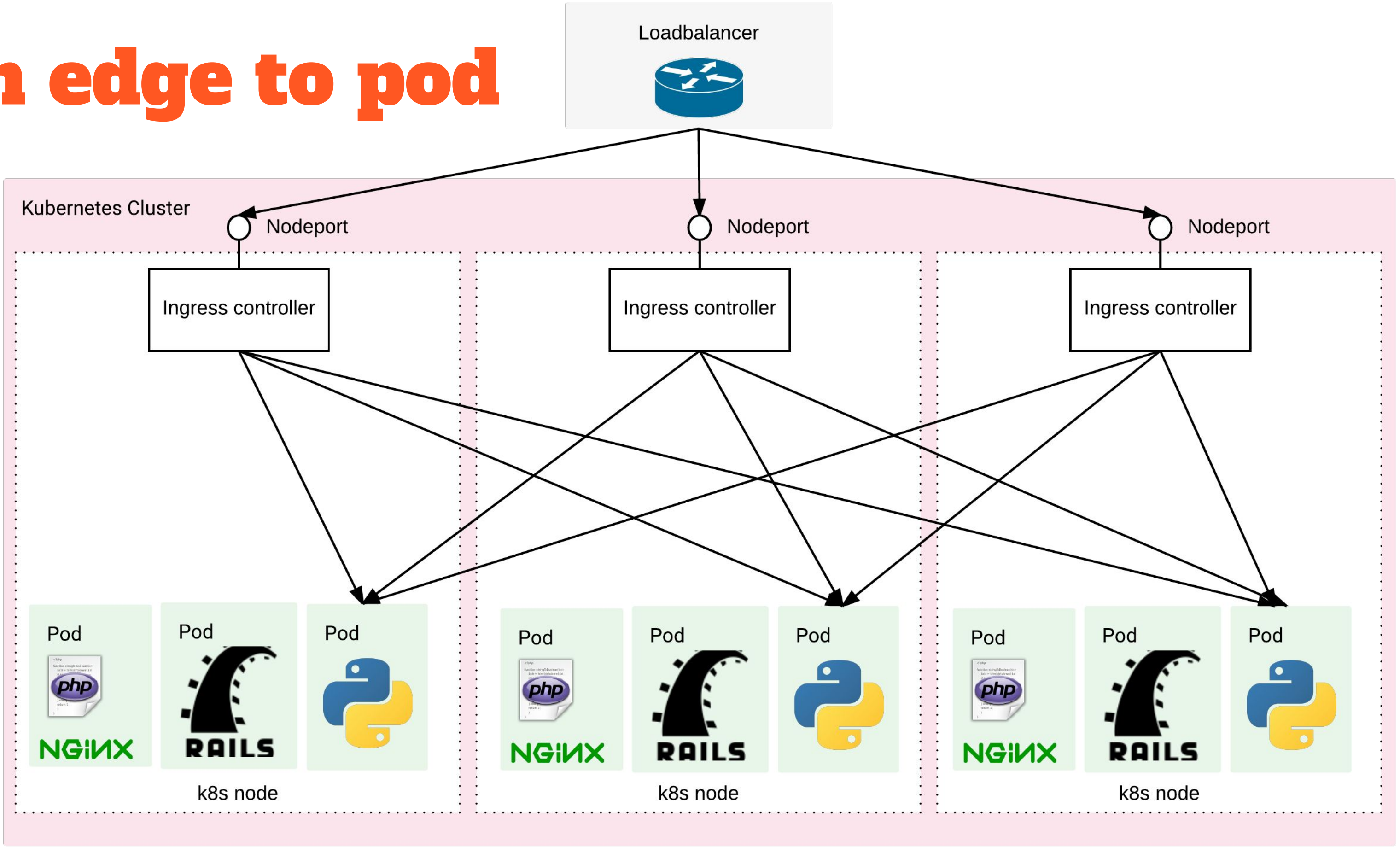
- OSSO uses on Ubuntu 16.04LTS. Saltstack managed.
  - Same as we use in other environments (Ubuntu/Debian), a lot of available experience.
  - Can just apply all our existing configuration management, monitoring, etc.
  - Has public feed for security CVE status (which we use for gocollect)
- Initially started with Container Linux (CoreOS)
  - It is nice but we dropped it.
- Kubernetes distributions are emerging
  - e.g. CoreOS has Tectonic, Redhat has OpenShift, etc.

Adding kubernetes to the mix

Run it next to your existing infrastructure and migrate your apps one by one (or even run them concurrently)



From edge to pod



K8s controllers

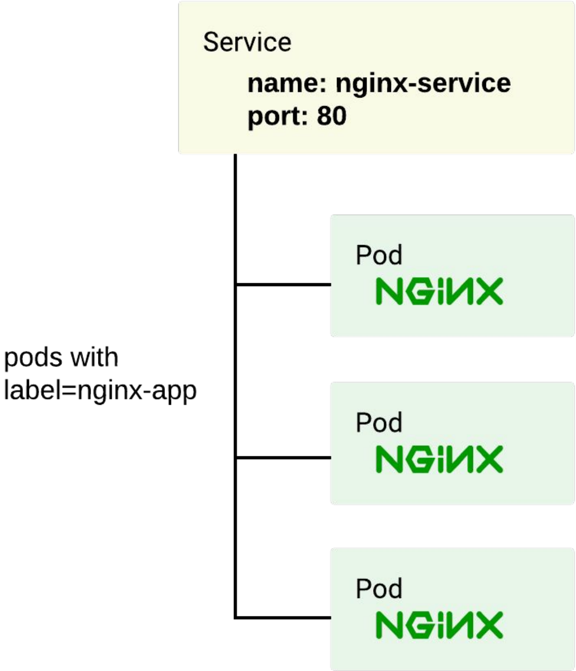
- **Replica Sets:** Controls how many pods are running
  - When to use: normally you use these indirectly through Deployments
- **Deployments**: Uses Replica Sets + deployment mechanics.
  - Use these for **stateless** applications (no persistent data etc).
- **StatefulSets:** For stateful services
  - Unique network identifiers, persistent storage, graceful deployment and scaling.
- **Daemon Sets:** For deploying a service on every node
  - For example ingress controllers, prometheus exporters, etc.

Services, kube-dns, kube-proxy

- The correct way to connect to containers
- No need to keep track of container ip’s
- Uses labels to target pods
- Map incoming port to a targetport inside a pod
- Works well together with kube-dns
- Service gets a virtual ip (which the dns will resolve to)
- Dns entry can be used from inside the application or from the ingress rules

Services, kube-dns, kube-proxy

```
apiVersion: v1
kind: Service
metadata:
 namespace: awesome
 name: nginx-service
spec:
 selector:
   app: nginx-app
 ports:
   - port: 80
```



Service

name: nginx-service
port: 80

pods with label=nginx-app

Pod

Pod

Pod

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
 namespace: awesome
 name: nginx-deployment
spec:
 replicas: 1
 template:
   metadata:
     labels:
       app: nginx-app
   spec:
     containers:
     - name: nginx-container
       image: ossobv/nginx:1.10.2
       ports:
       - containerPort: 80
```

k8s ConfigMap and Secrets

- The right place to store your configuration parameters
- Secrets and ConfigMaps work similar

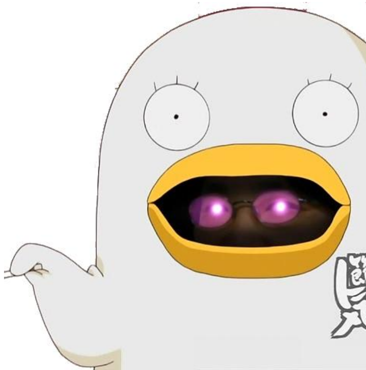
```
apiVersion: v1
data:
 tls.crt: #BASE64_DATA#
 tls.key: #BASE64_DATA#
kind: Secret
metadata:
 annotations:
   kubernetes.io/tls-acme: "true"
 creationTimestamp: 2017-04-10T13:02:11Z
 name: awesome-nginx-tls
 namespace: awesome
 resourceVersion: "605871"
 selfLink: /api/v1/namespaces/awesome/secrets/awesome-nginx-tls
 uid: e935817d-1ded-11e7-9fad-0cc47aeabeb1
type: kubernetes.io/tls
```

```
apiVersion: v1
data:
 default: |
   server {
     listen 80 default_server;
     root /var/www/html;
     index index.html;
     server_name _;
     location /healthz {
        return 200;
     }
     location / {
        try_files $uri $uri/ =404;
     }
   }
kind: ConfigMap
metadata:
 name: nginx-static-html-config
 namespace: awesome
```

k8s.osso.ninja

Grab your ninja and deploy

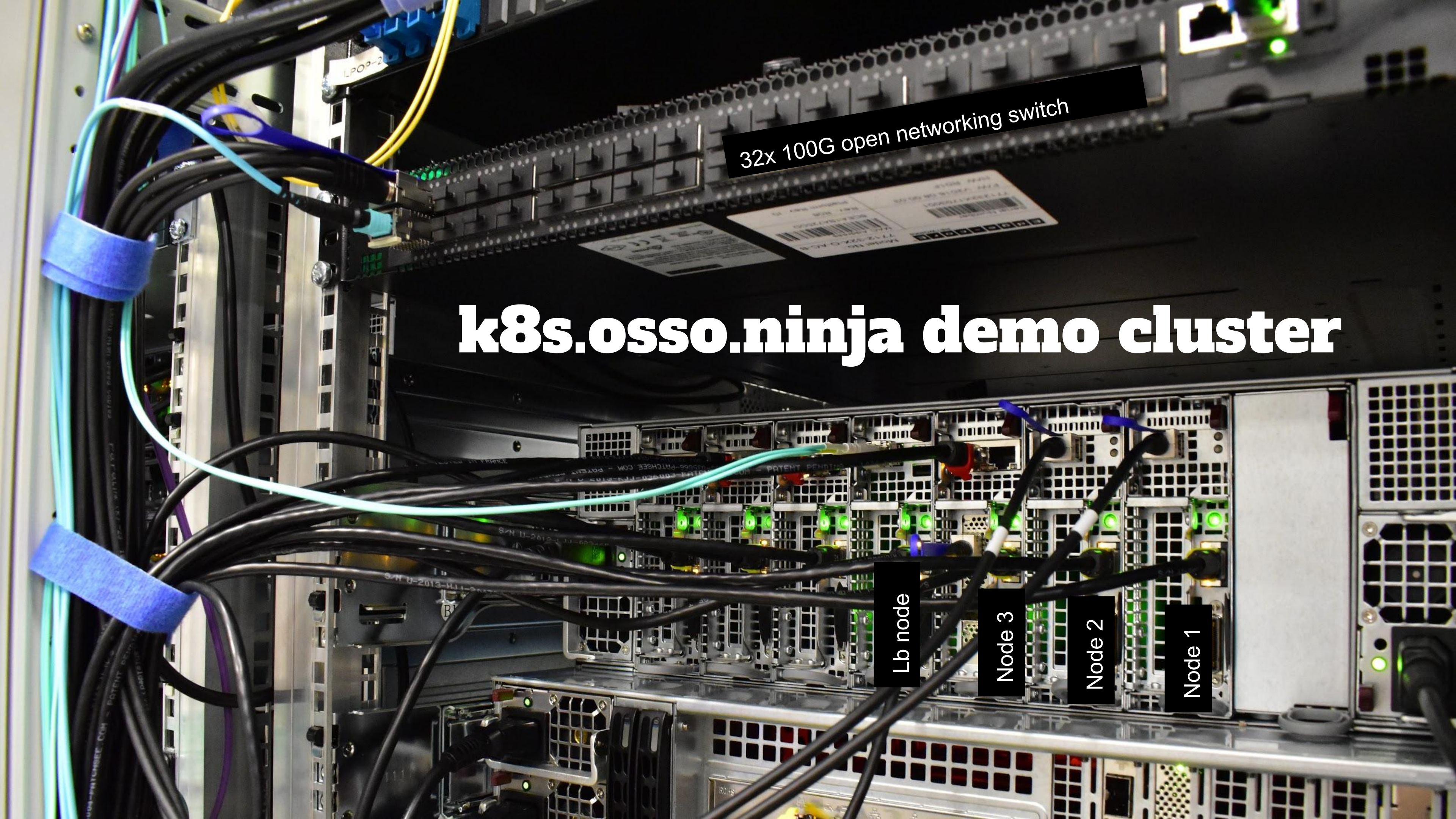
Getting started instructions:
https://get.osso.ninja



Cluster k8s node specs

- E5-2630v4
- 128GB RAM
- 25Gbit/s network
- 240GB ssd raid1

k8s.osso.ninja demo cluster



32x 100G open networking switch

Lb node

Node 3

Node 2

Node 1

Protips

- Avoid shared file systems.
- Focus on getting your app deployments to K8s first.
  - Make then stateless if needed.
- Leave databases, legacy apps and other persistent data services for later.
  - And then use StatefulSets
- Run k8s cluster next to your current servers
  - Migrate apps/services one by one.
- Play around and get some experience
- Read the docs and ask questions

Q&A