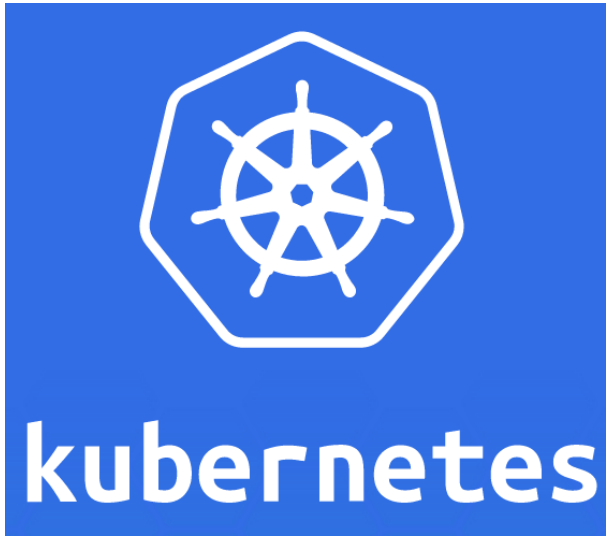
Kubernetes 101 for Python Programmers

https://github.com/bigbitbus/k8s-tutorial-python



About Me

BigBitBus

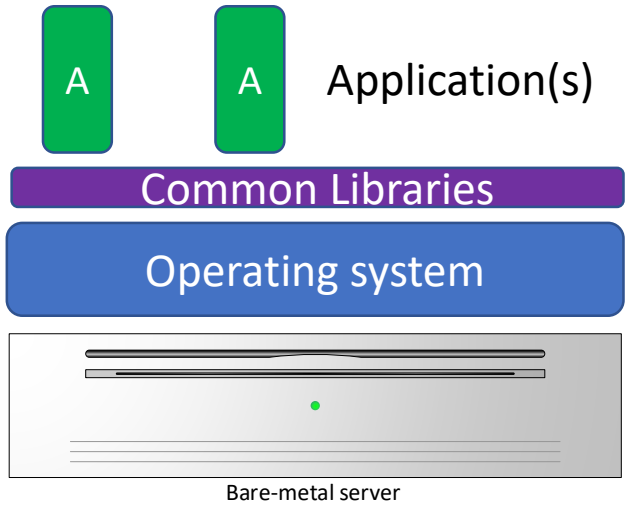
Transparency in public cloud and big data/analytics

https://www.linkedin.com/in/sachinkagarwal/

Bare-metal, Virtual Machines, Containers

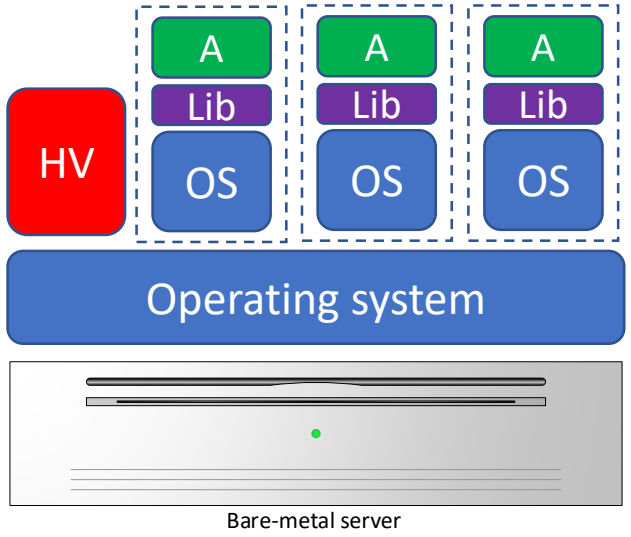
Bare Metal

- Weak encapsulation
- Long install/start-up times
- Ideal for single application e.g. a Hadoop node
- Hard to right-size



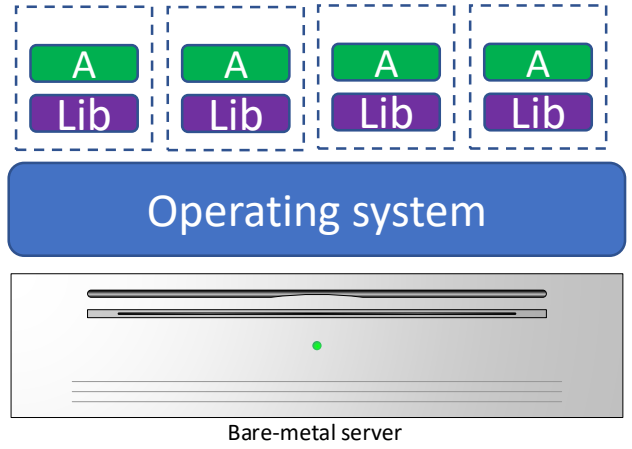
Virtual machines

- Strong encapsulation
- Hardware emulation overhead (per VM)
- Per-VM operating system overhead
- Long install/start-up times



Containers

- Encapsulation *lite*
- Small overhead – only single kernel running
- Smaller deployment artifacts expressed as code
- Quick start-up times

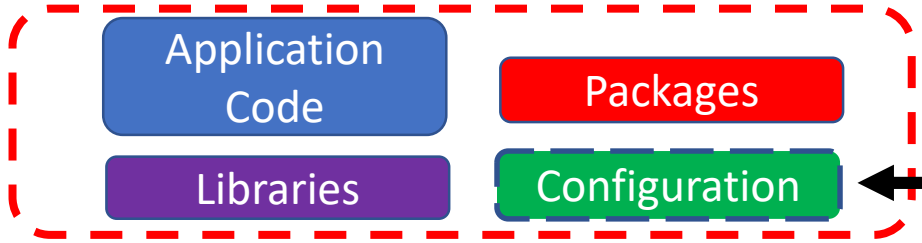


Docker – container runtime

Build

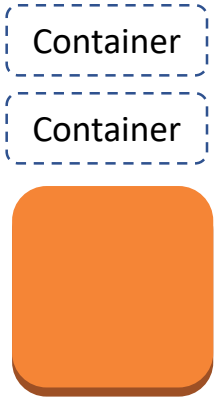
Run

- *Download immutable image*
- *Configure and run*



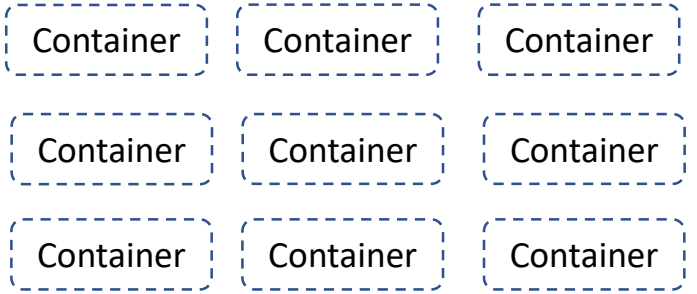
Inject environment and secrets at runtime

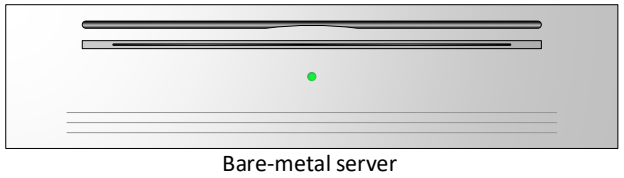
Beyond Single Node Docker

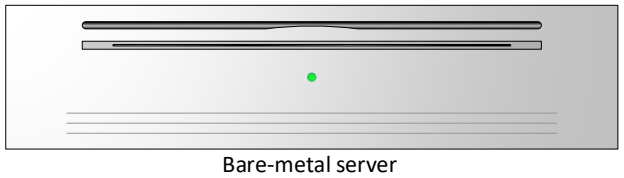


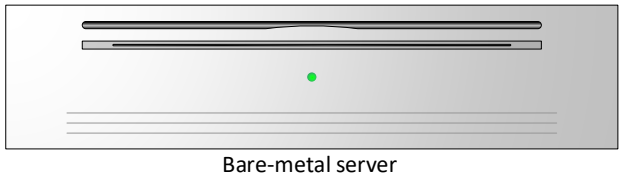
Single node

Kubernetes cluster




Bare-metal server


Bare-metal server

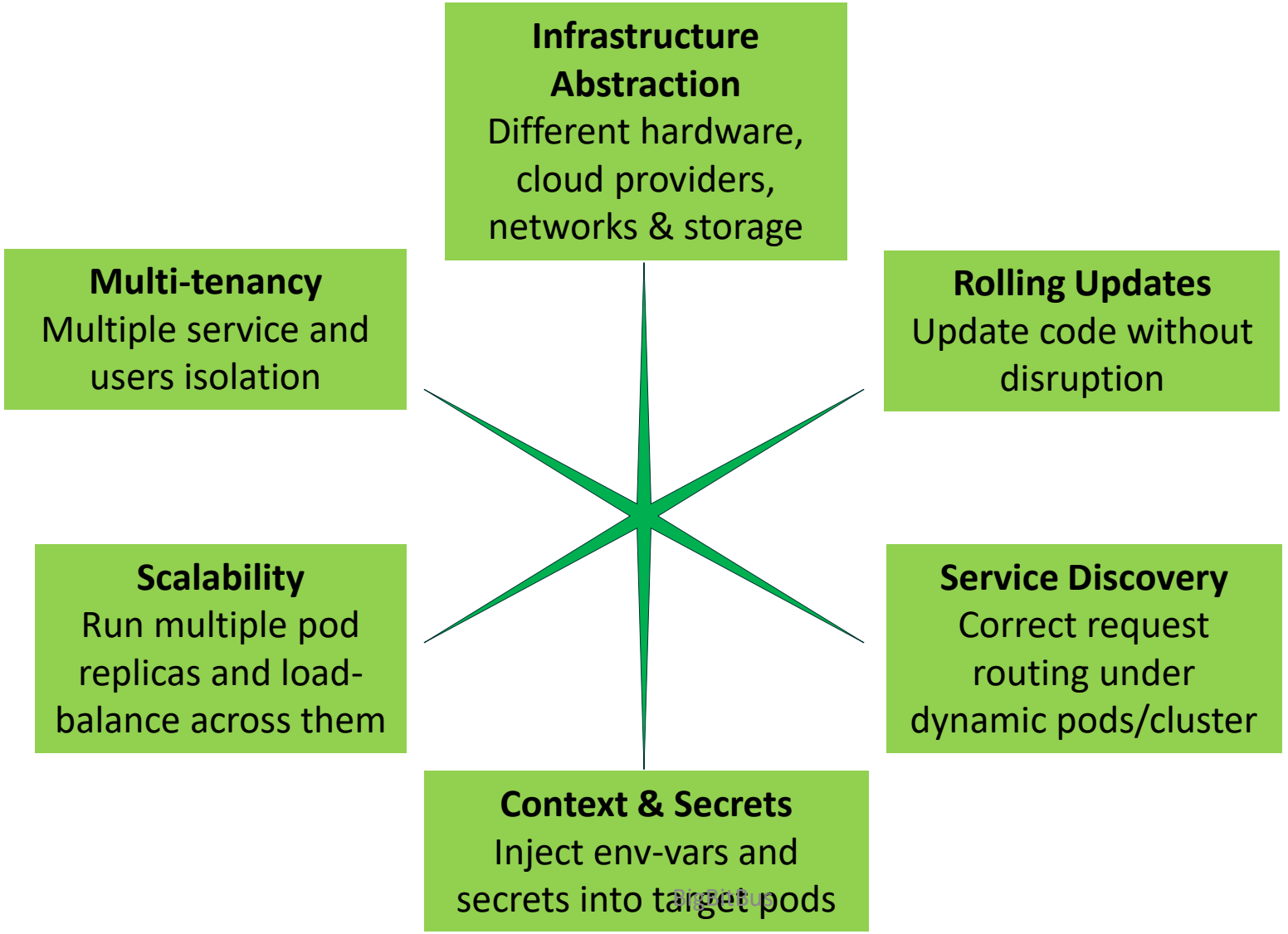

Bare-metal server

Multi-node production scale-out

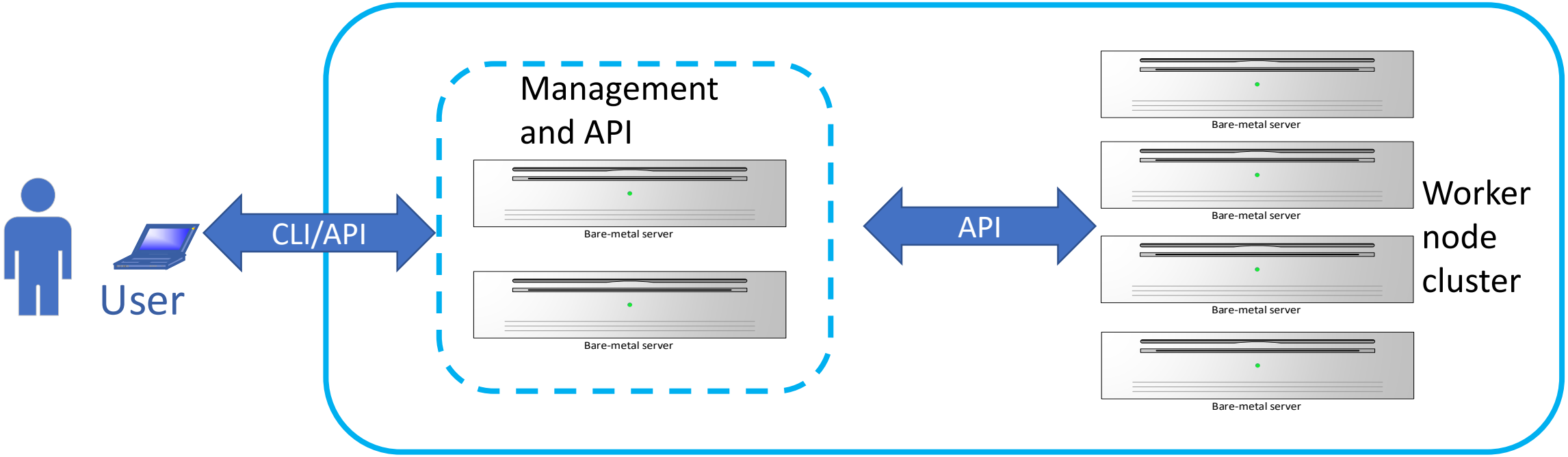
1. High-availability
2. Horizontal scalability

EC2
VM instance

Beyond Manual Containers



Kubernetes Cluster



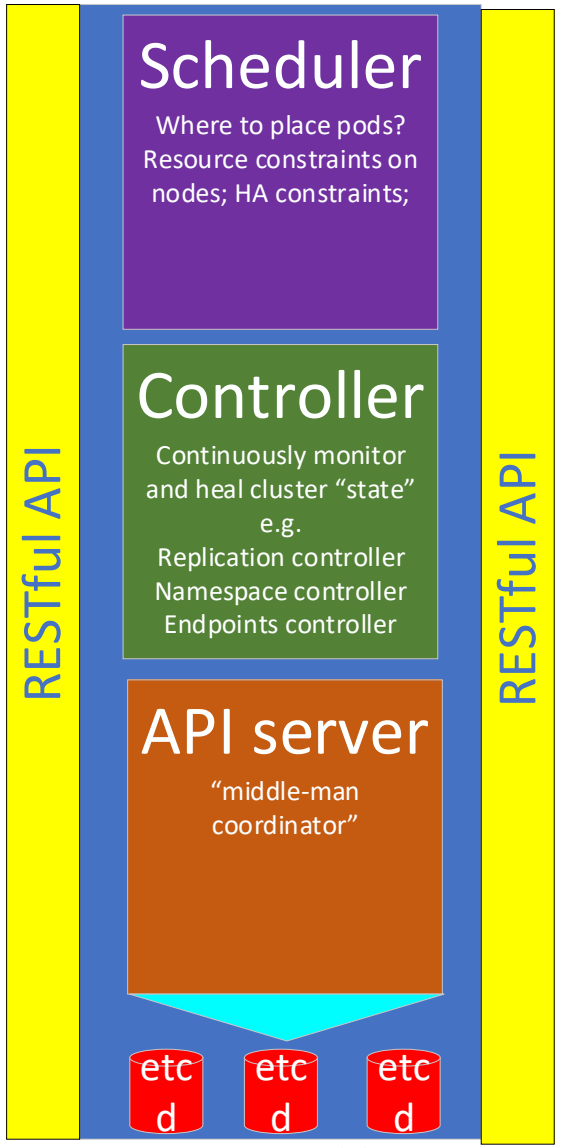
External services

Kubernetes Architecture

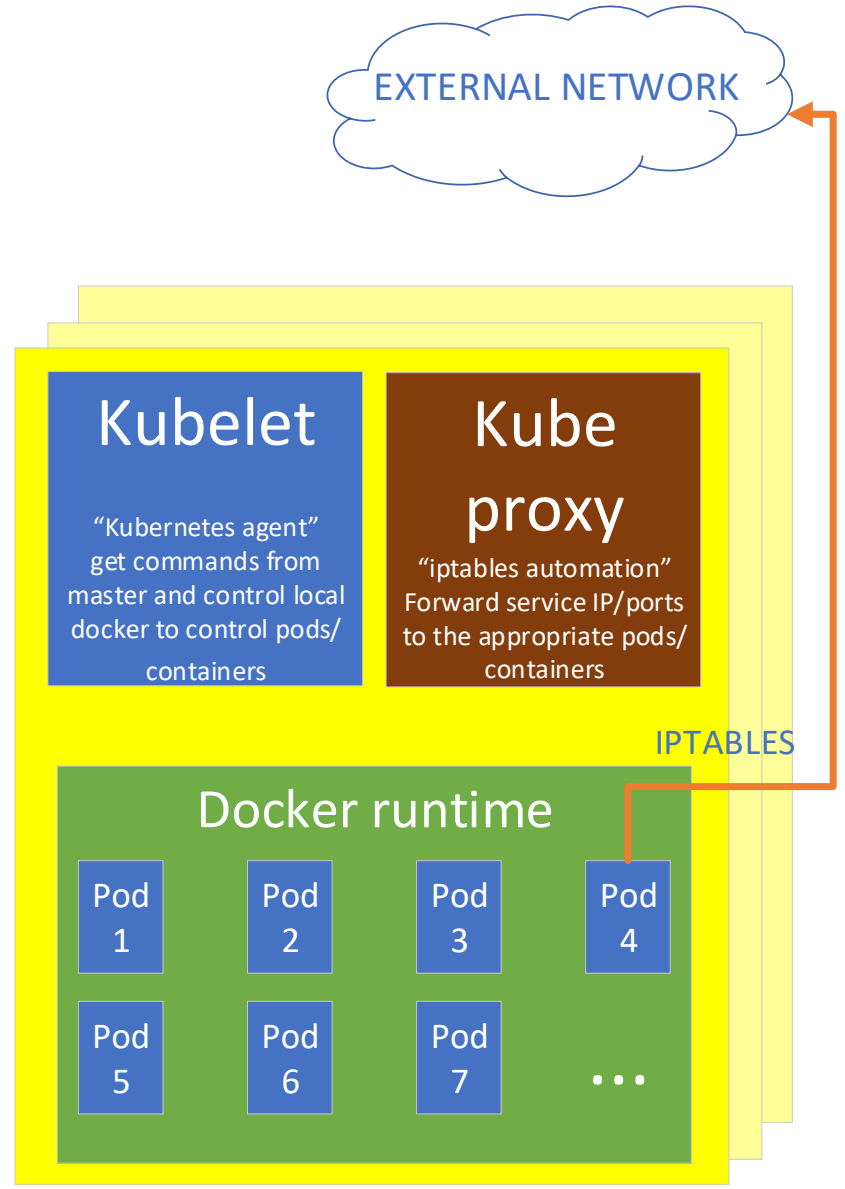
Multiple components

- Management plane
- Worker plane

Worker nodes can be bare-metal or virtual machines



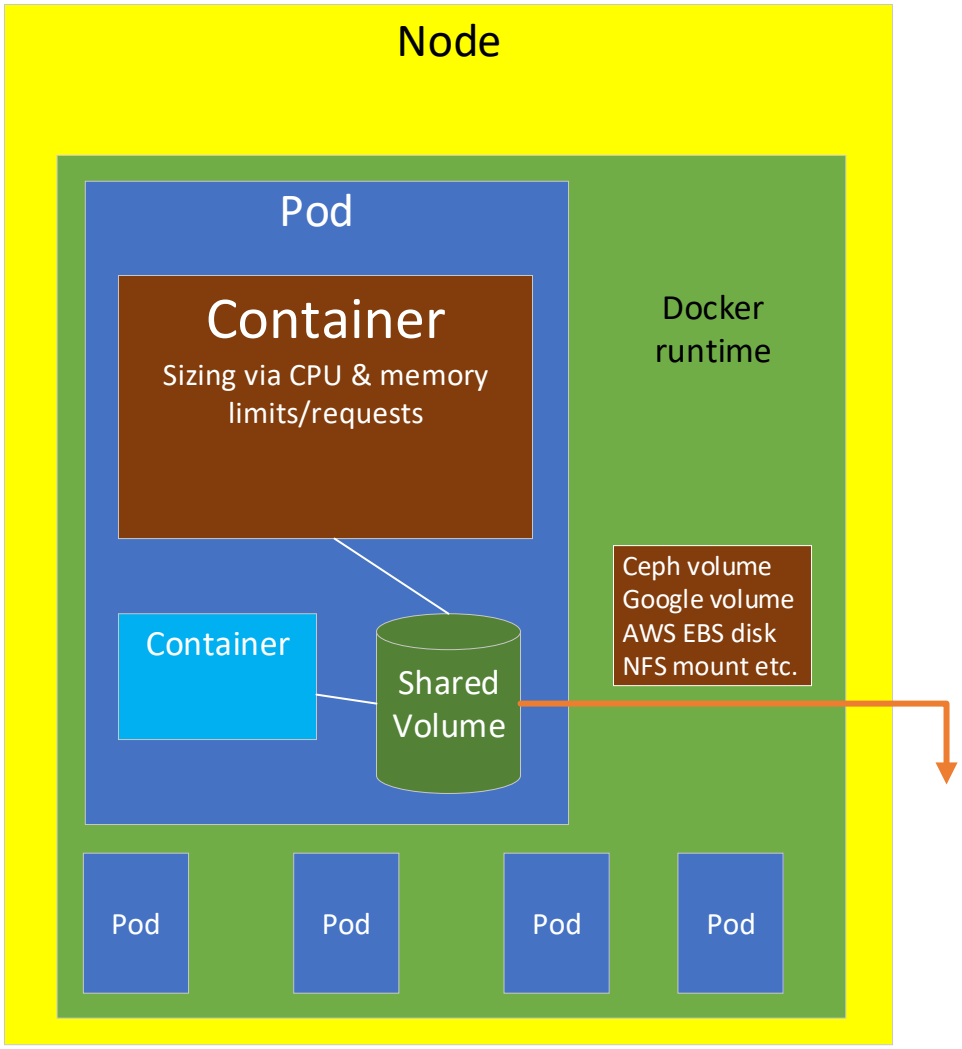
Master(s) management plane



Nodes – servers running Kubernetes workers

Pod

- The smallest independent “unit” of infrastructure in K8s
  - Scaling/redundancy
- A logical “host”
- Multiple containers & volumes
- Localhost and IPC connectivity
- Cannot span nodes



Application server + log watcher

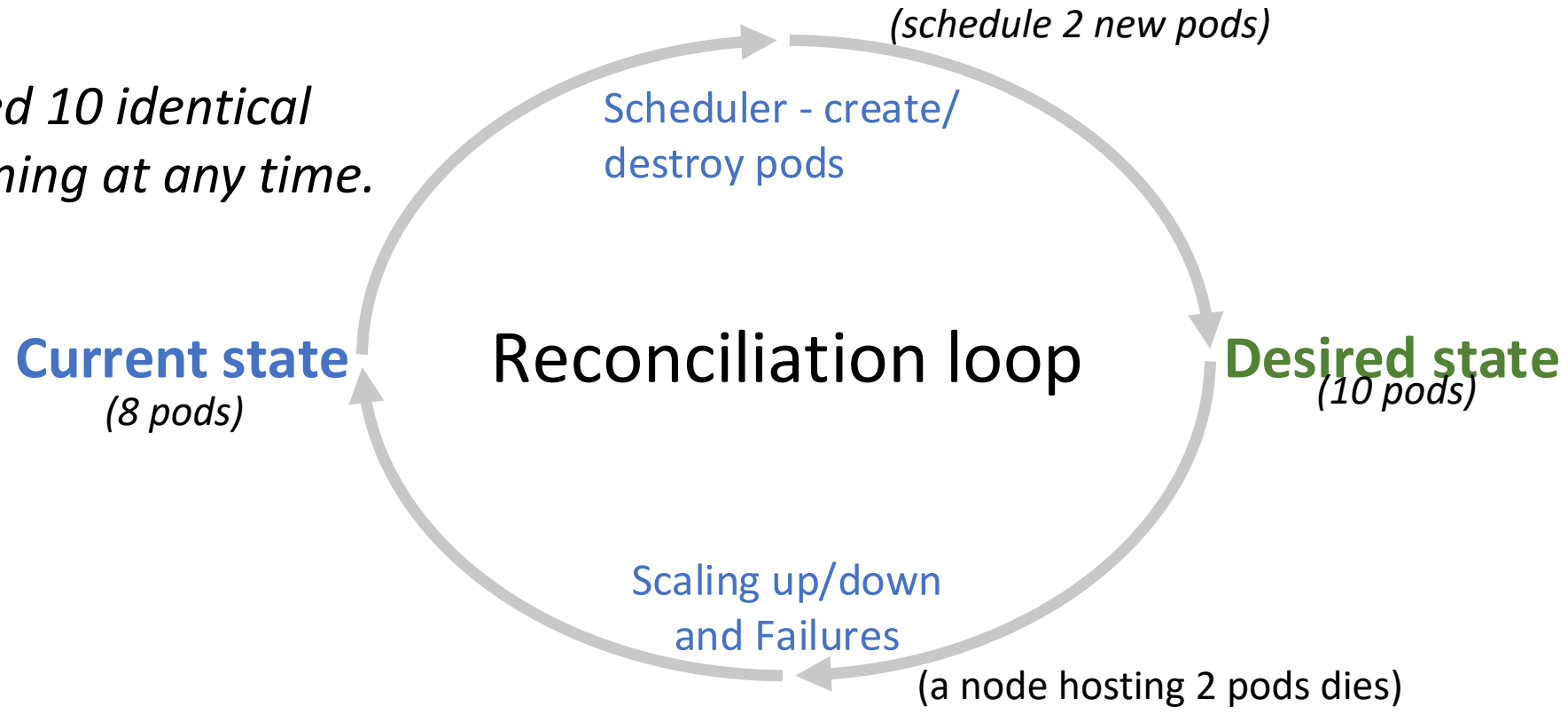
Application server + local cache

https://kubernetes.io/docs/concepts/workloads/pods/pod/

Replicasets (of pods)

- Keep a defined number of identical pod replicas running in the Kubernetes cluster

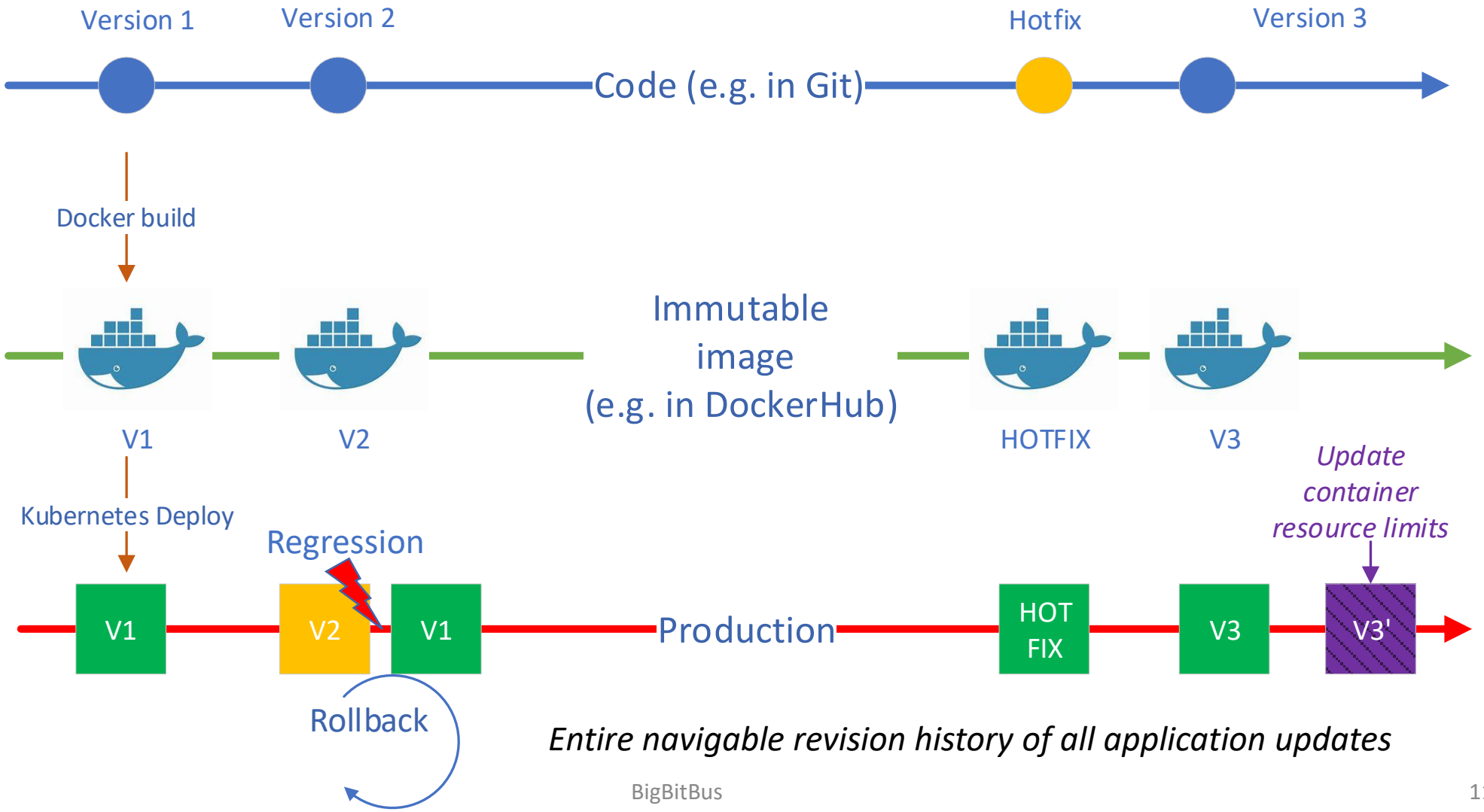
Example: We need 10 identical Django pods running at any time.



https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

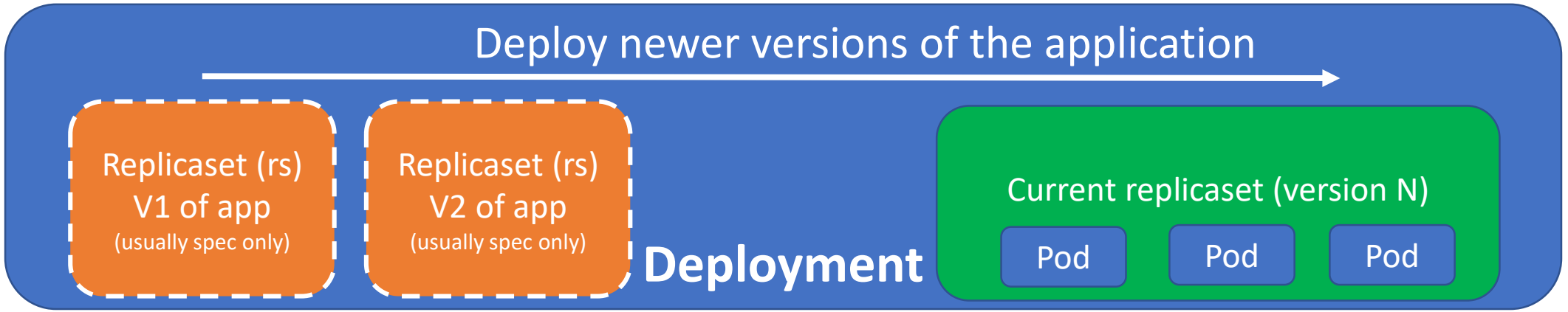
Application updates, rollbacks, jumps to any revision

Deployments



Entire navigable revision history of all application updates

Deployment defined in YAML



- Deployments help manage application versions
  - Click here to see the Django deployment yaml file
  - Click here to see the Postgres deployment yaml file
- A new replicaset is created every time the deployment changes
- Deleting deployments will delete the underlying replica-sets and pods

Services

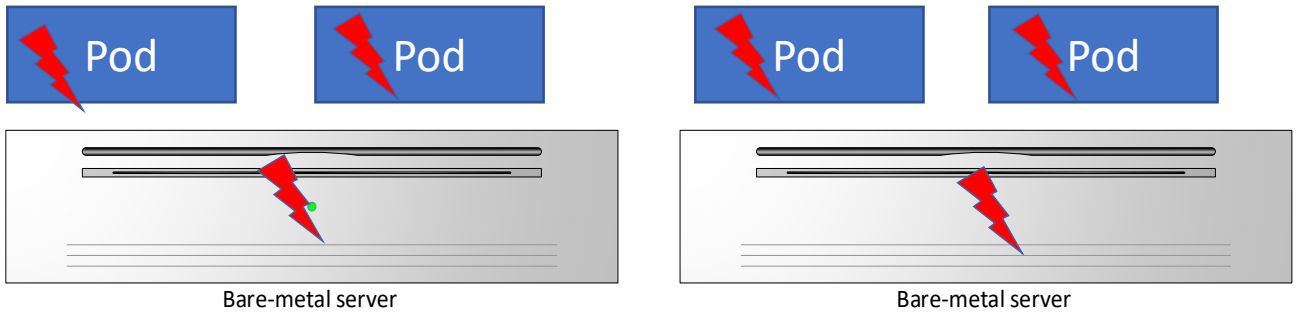
Services route traffic to the appropriate pods within the k8s cluster

Service Flavors

1. Cluster-IP: a service to which other services within the cluster connect. Example
2. Load balancer: Expose service externally (Load-balancer). Example
3. External Name: Connect to an external service. Example

Pods are "cattle" - expendable, replaceable - not pets

Underlying pods and hardware are ephemeral and very dynamic



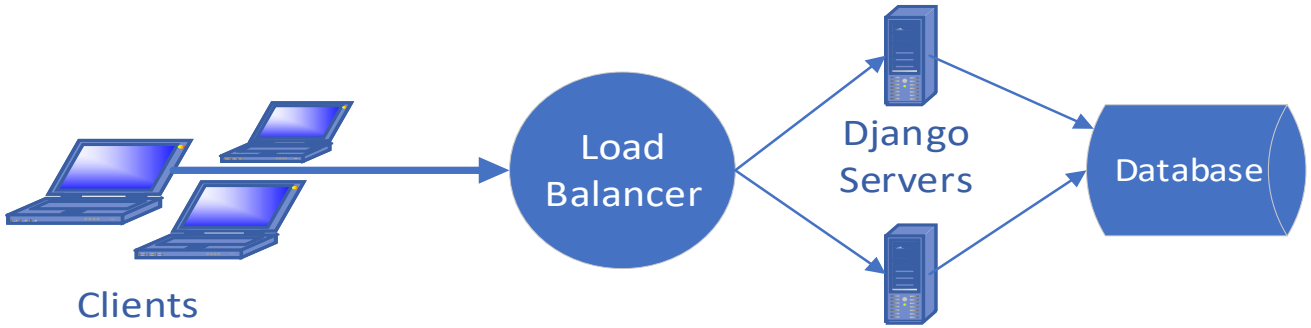
Environment Variables & Secrets

- Inject meta-data into pods, e.g. based on namespace
- Configmaps for environment variables
  - Example (development & production namespace)
- Secrets
  - Create a secret

```
kubectl create secret generic db-password --from-literal=db-password=superSecret
```

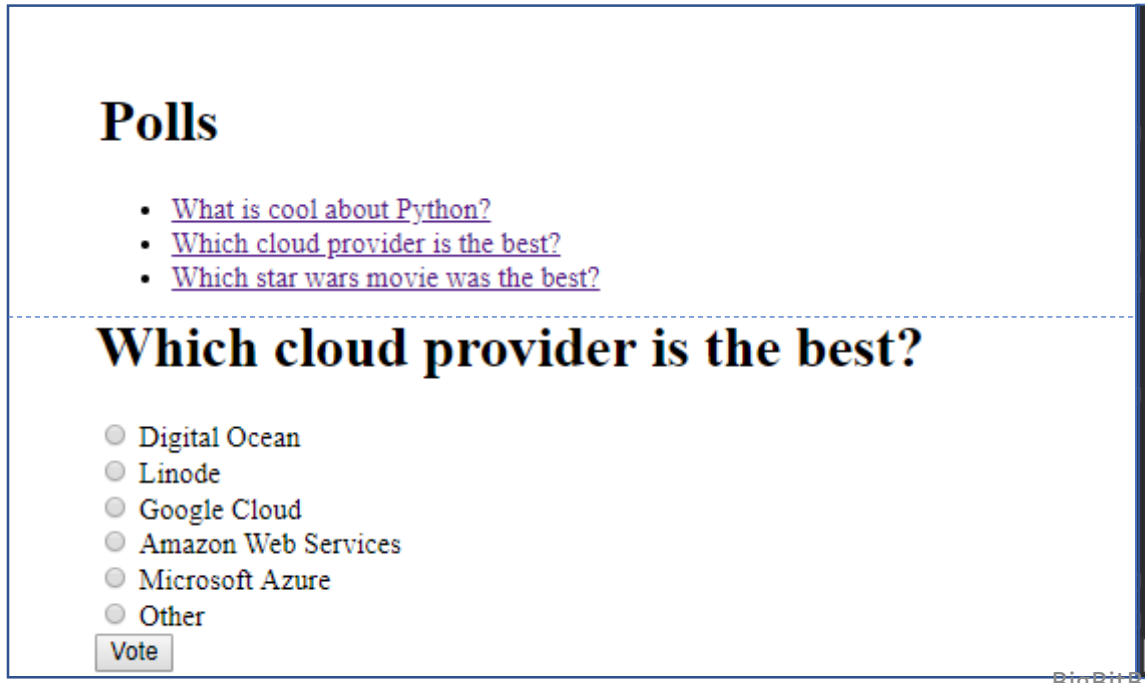
  - Expose secret inside pod as an environment variable

Polls Web Application Example



- Stateless Django servers (cattle)
- All state data stored in database (pet)

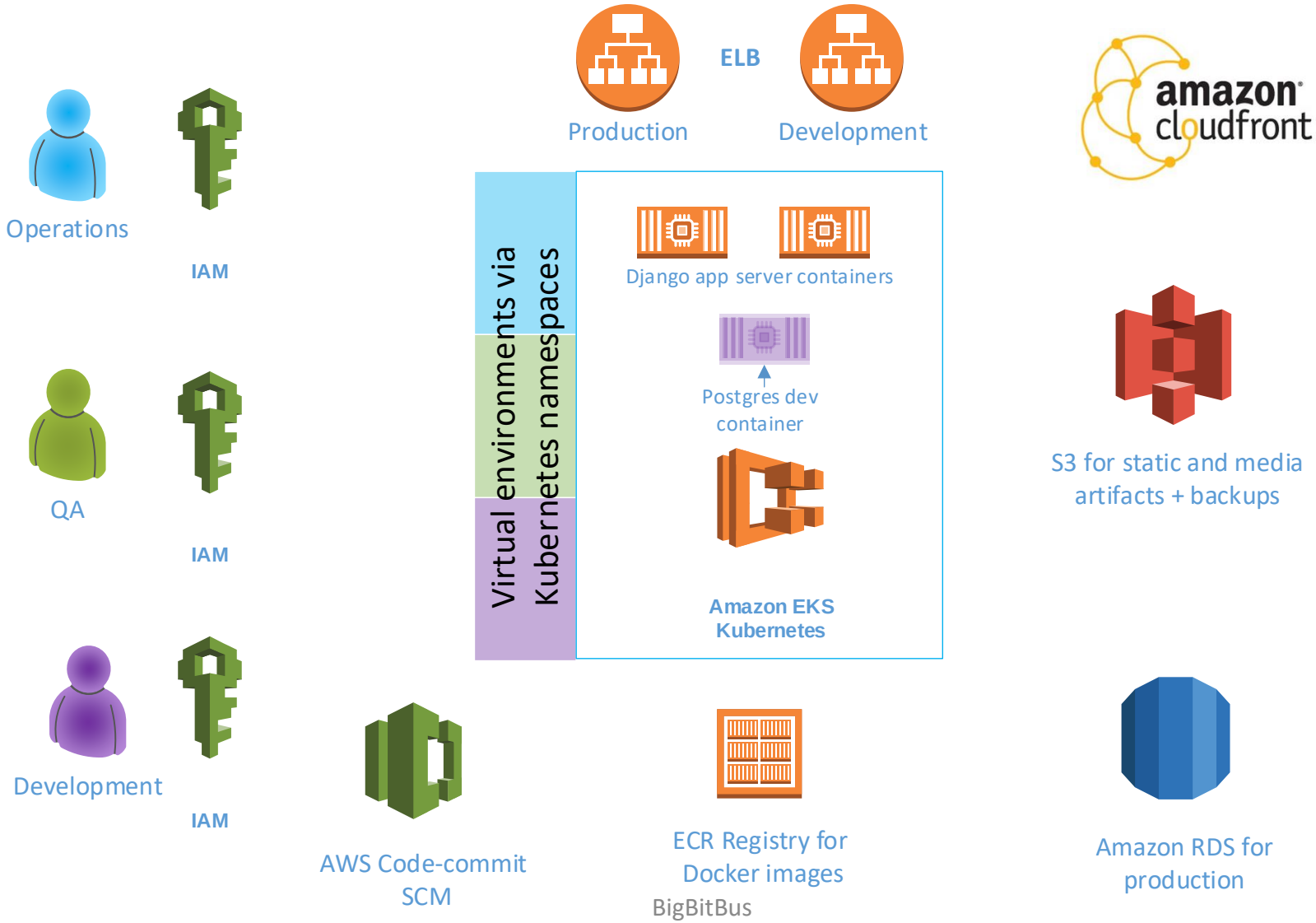
Polling Application in Browser



State definition - models.py

```python
from django.db import models


class Poll(models.Model):
    question = models.CharField(max_length=200)

    def __str__(self):              # Python 3: def __unicode__(self):
        return self.question


class Choice(models.Model):
    poll = models.ForeignKey(Poll, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):              # Python 3: def __unicode__(self):
        return self.choice_text
```

Architecture



Demo

https://github.com/bigbitbus/k8s-tutorial-python

Explore Further

- Labels and Annotations: Very powerful construct to filter and track K8s resources: Required reading!
- Horizontal pod autoscaling: Triggers for pod quantity adjustment
- Statefulsets (instead of replicasets): Customizing individual pods in a deployment e.g. to take on different roles.
- Persistent volumes: Attach block devices to pods
- Jobs and Cronjobs: Running one-off tasks or periodic pods.

Appendix and backup slides

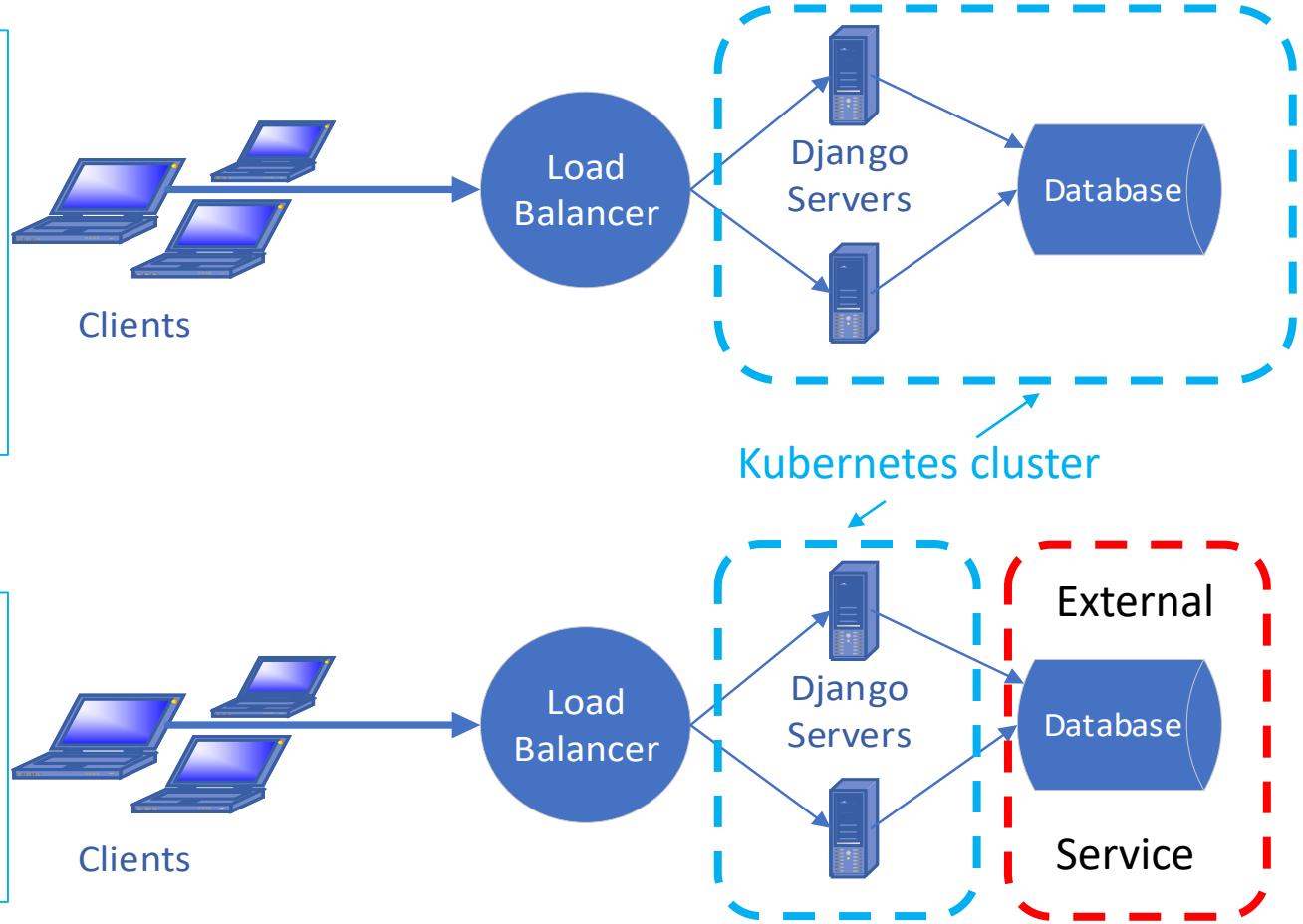
Stateful Databases in Kubernetes (?)

Option 1

- Kubernetes provides constructs for this (stateful sets, volumes, etc.)
- Possible cost saving (bin packing containers)
- Is self-managed state worth your developer/SRE time?

Option 2

- Use cloud-provider stateful solutions
  - AWS RDS, Google cloudsql, etc.
- Expensive, but much less operations tasks



Cost Math

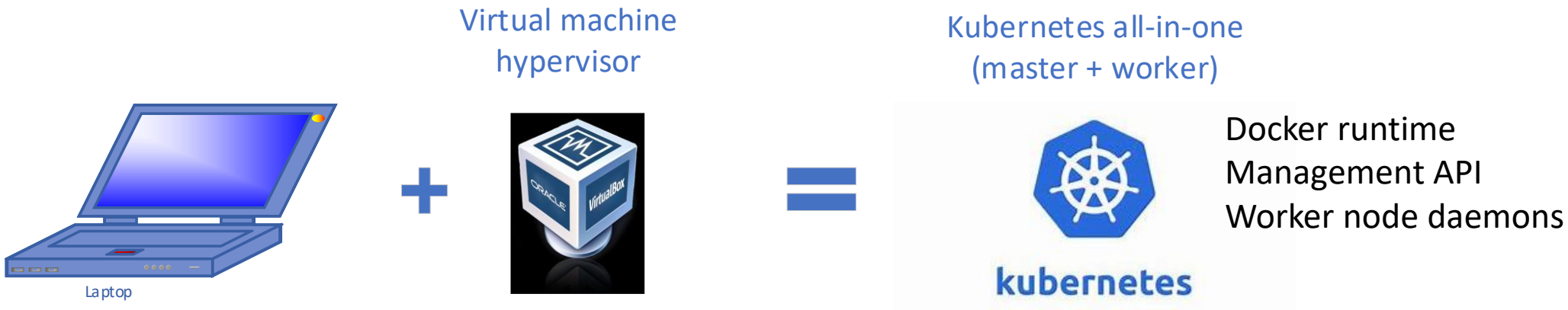
Cloud provider

- Master API server Currently (11/2018) $0.20/hour per Amazon EKS cluster **($1,752** annually)
- Add worker node VMs cost
- You still need to buy all the storage/load balancers/IP addresses etc.

Minikube

- Free, but needs a good PC capable of running a fairly big VM.
- No high availability - this is only for development work

Minikube: Kubernetes on your Laptop



- Installation instructions for Windows, Mac and Linux
  https://kubernetes.io/docs/tasks/tools/install-minikube/

- Default: 2 cores + 2 GB RAM = capable of running a few containers

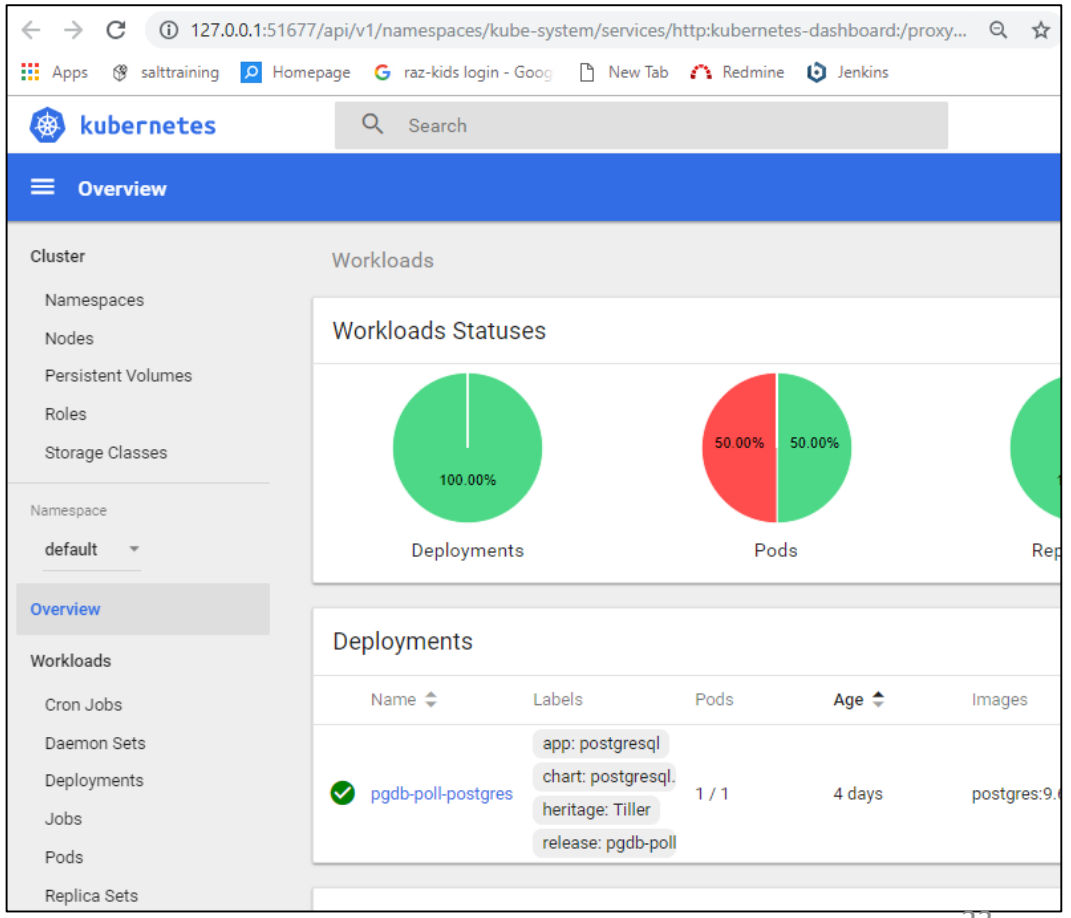
Starting minikube

minikube cli

```
# start minikube, launch VM
$ minikube start
# check minikube status
$ minikube status
# start minikube GUI
$ minikube dashboard
```

```
capacity:
  cpu: "2"
  ephemeral-storage: 16888216Ki
  hugepages-2Mi: "0"
  memory: 2038624Ki
  pods: "110"
```

Dashboard
General-purpose web UI for Kubernetes clusters



Kubectl – Kubernetes CLI **`kubectl --help`**

- Wrapper around RESTful API: GET, CREATE, DEPLOY etc. action-words

```
kubectl get nodes
kubectl get nodes -o wide
kubectl get node minikube -o yaml
```

More information, yaml or json

```
kubectl config get-contexts
kubectl config use-context aws
kubectl get nodes
```

The same kubectl can point to multiple k8s clusters

```
kubectl create namespace development
kubectl get namespaces
kubectl --namespace development get pods
```

Virtual k8s cluster via namespaces

Run a one-off Pod

```
kubectl run my-shell --rm -i --tty --image ubuntu -- bash
```

```
ubuntu@ip-172-31-45-221:~$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
my-shell-68974bb7f7-tbfpx   1/1     Running   0          1m
ubuntu@ip-172-31-45-221:~$ kubectl get deployments
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-shell   1         1         1            1           1m
ubuntu@ip-172-31-45-221:~$ kubectl delete deployment my-shell
deployment.extensions "my-shell" deleted
ubuntu@ip-172-31-45-221:~$ kubectl get pods
No resources found.
```

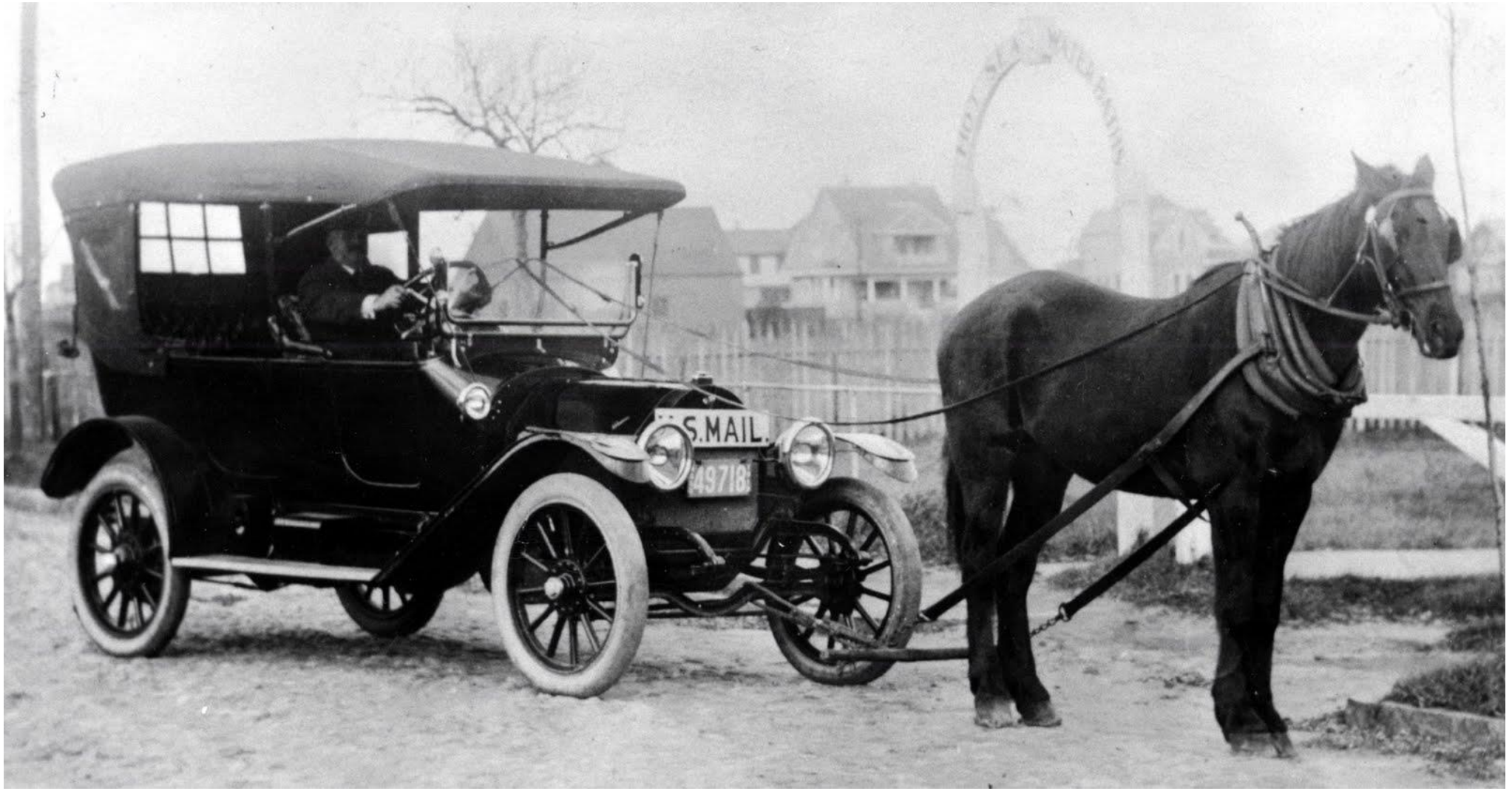
A pod is created, a
A deployment is also created
Cascading delete – pod also deleted

Inside the pod
Just like a “normal” Linux box

```
root@my-shell-68974bb7f7-tbfpx:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
172.17.0.5      my-shell-68974bb7f7-tbfpx
root@my-shell-68974bb7f7-tbfpx:/#
```

Kubernetes Anti-patterns

- Large pods – avoid making pets!
- Imperative kubectl commands instead of declarative yaml
- Single namespace for multiple environments
- Lack of consistent names (images/deployments/services etc.)
- Single cluster for development and production
- Missing application-level health-checks
- Missing cluster-level (node) auto-scaling



Running one-off pods is not the Kubernetes use-case