

# Kubernetes 101 for Python Programmers

*Learn the ABCs of Kubernetes and how to get started on using managed containers for your python development on your local machine and also for production deployments on a cloud provider-managed Kubernetes cluster.*

**Sachin Agarwal <sachin@bigbitbus.com>**

Download the latest version of these slides from  
<https://github.com/bigbitbus/k8s-tutorial-python>

**DRAFT v.08**

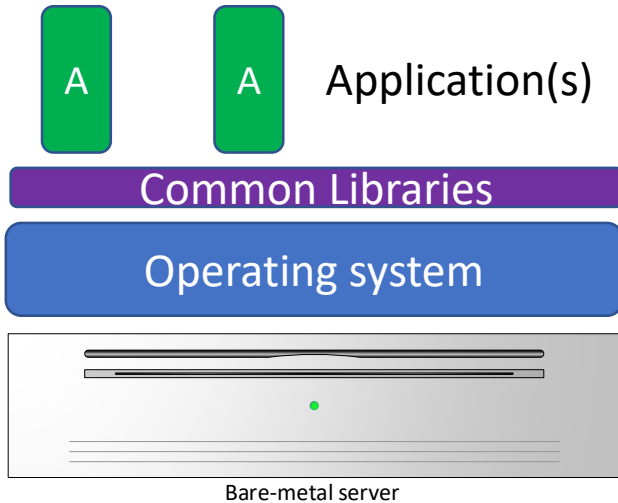
# Tutorial Agenda

1. Background
  - Docker container runtime
2. Kubernetes Architecture & Components
3. Kubernetes for Managed Containers
  - Replicasets, services, deployments, configmaps and secrets
  - Django application with K8s – scaling, updating code
4. Further reading

# Bare-metal, Virtual Machines, Containers

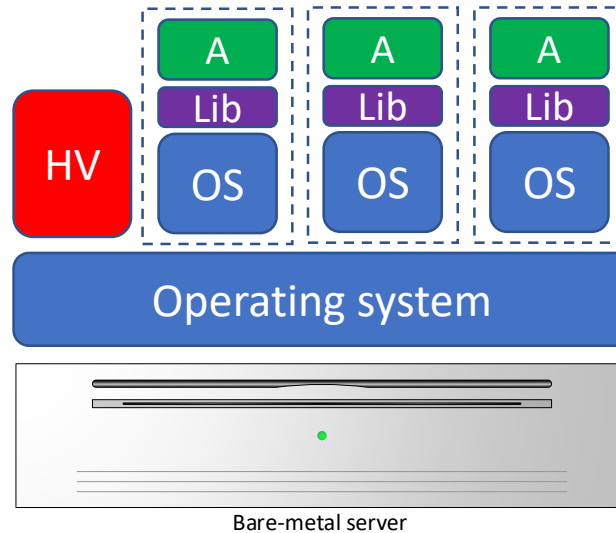
## Bare Metal

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



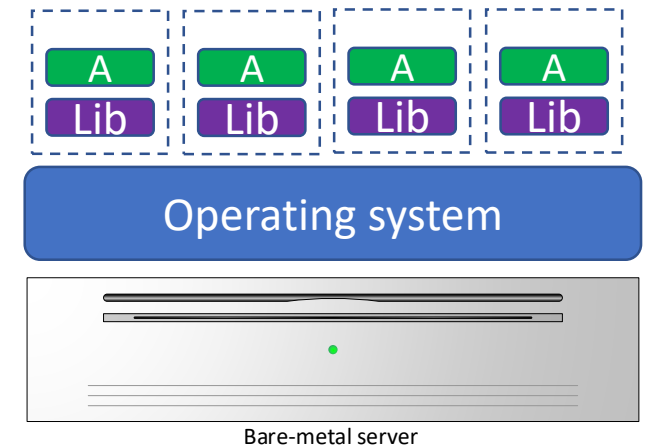
## Virtual machines

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



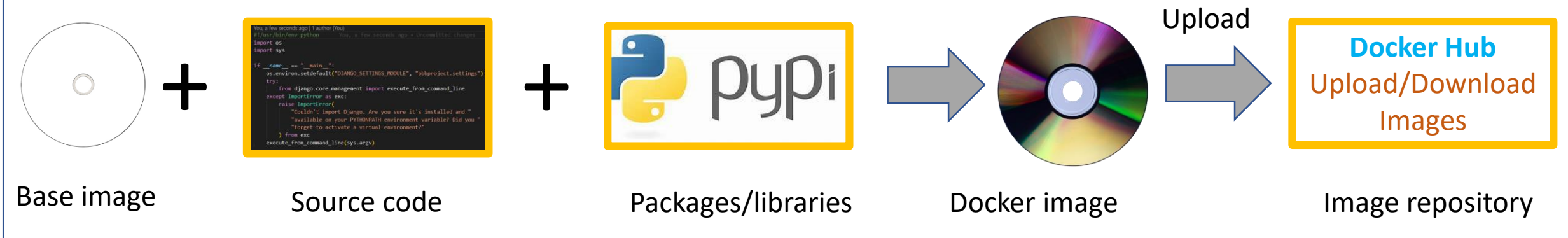
## Containers

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



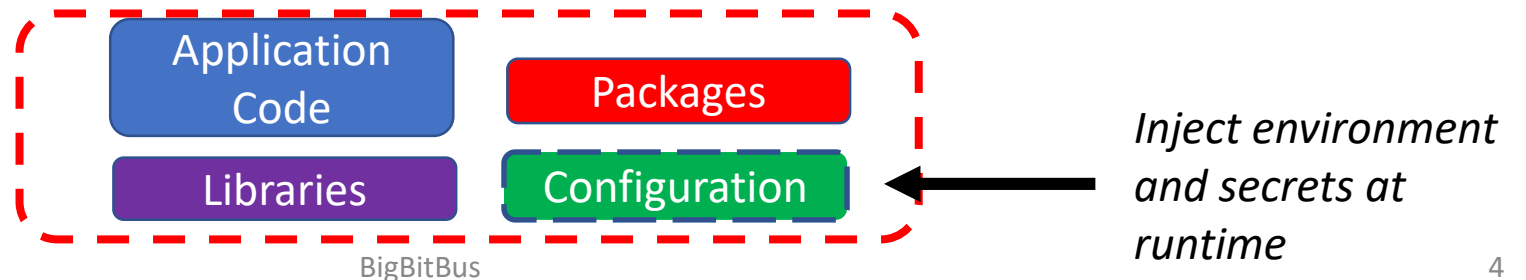
# Docker – container runtime

## Build



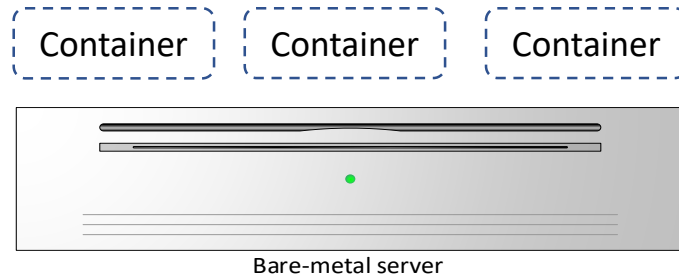
## Run

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

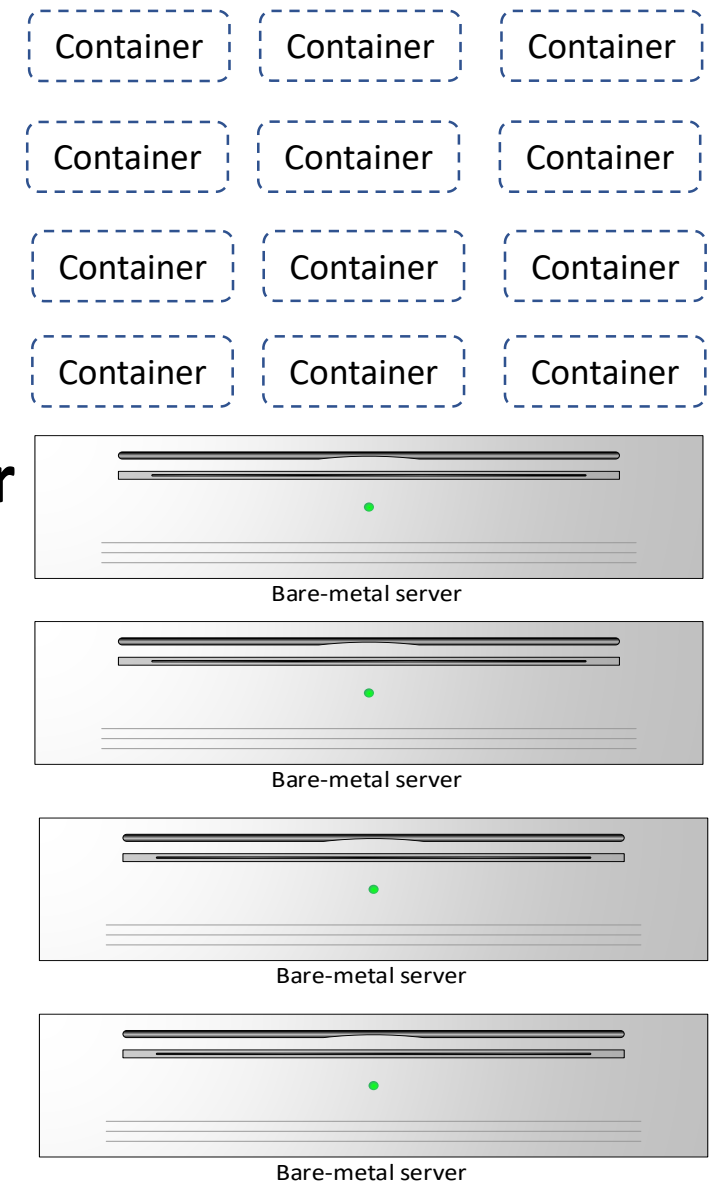


# Beyond Single Node Docker

Single node



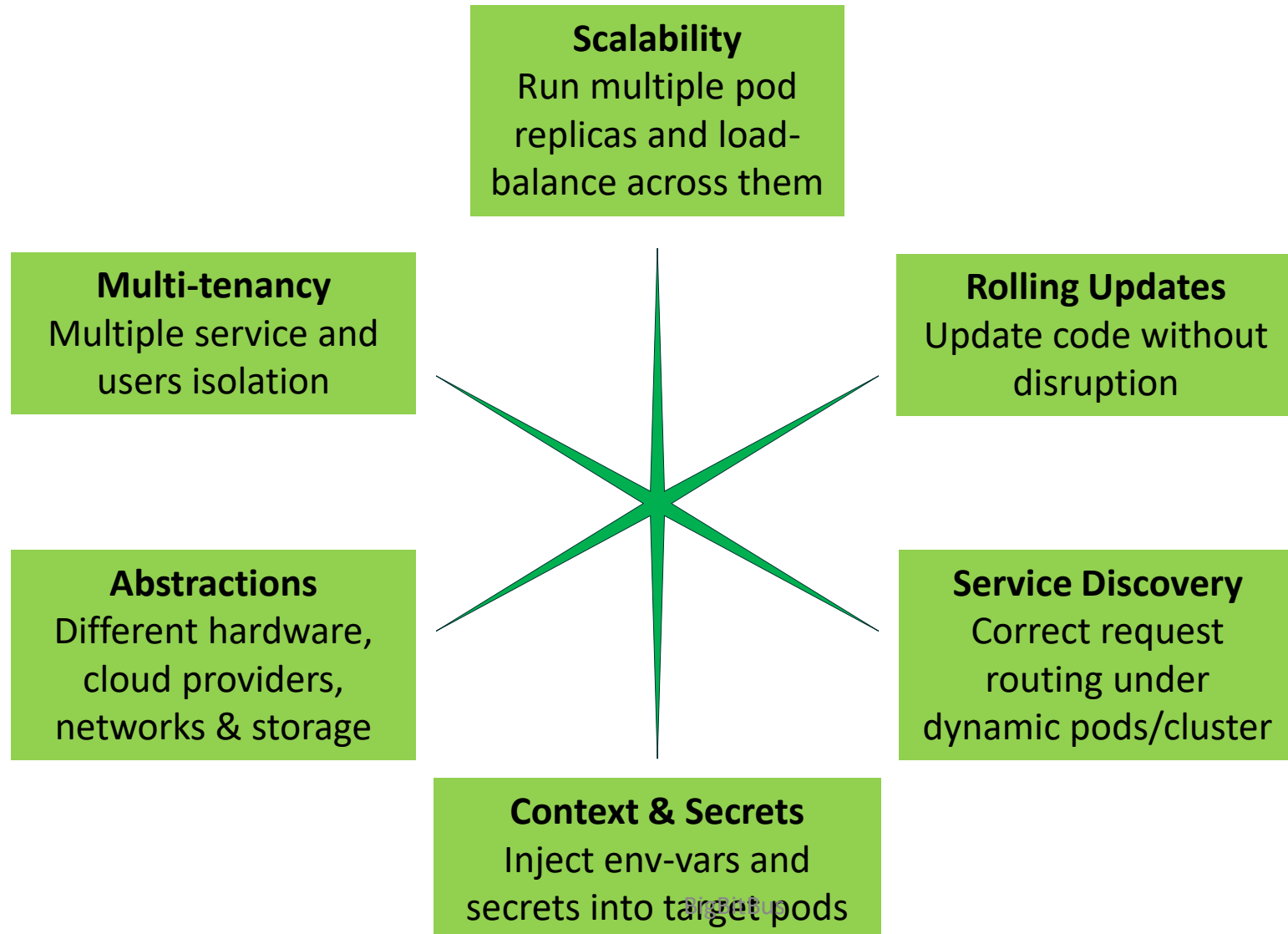
Node cluster



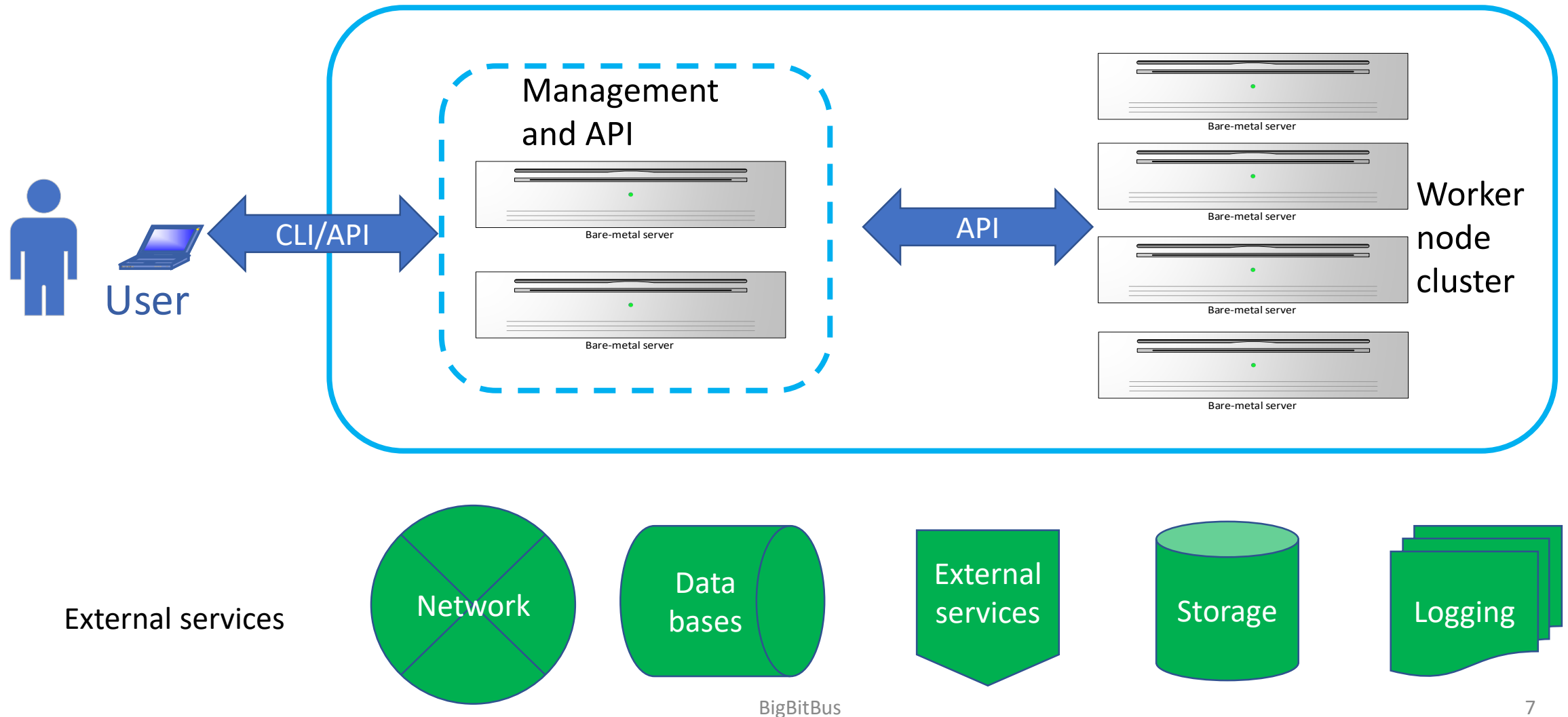
*Multi-node production scale-out?*

1. *High-availability*
2. *Horizontal scalability*

# Beyond Manual Containers



# Kubernetes Cluster

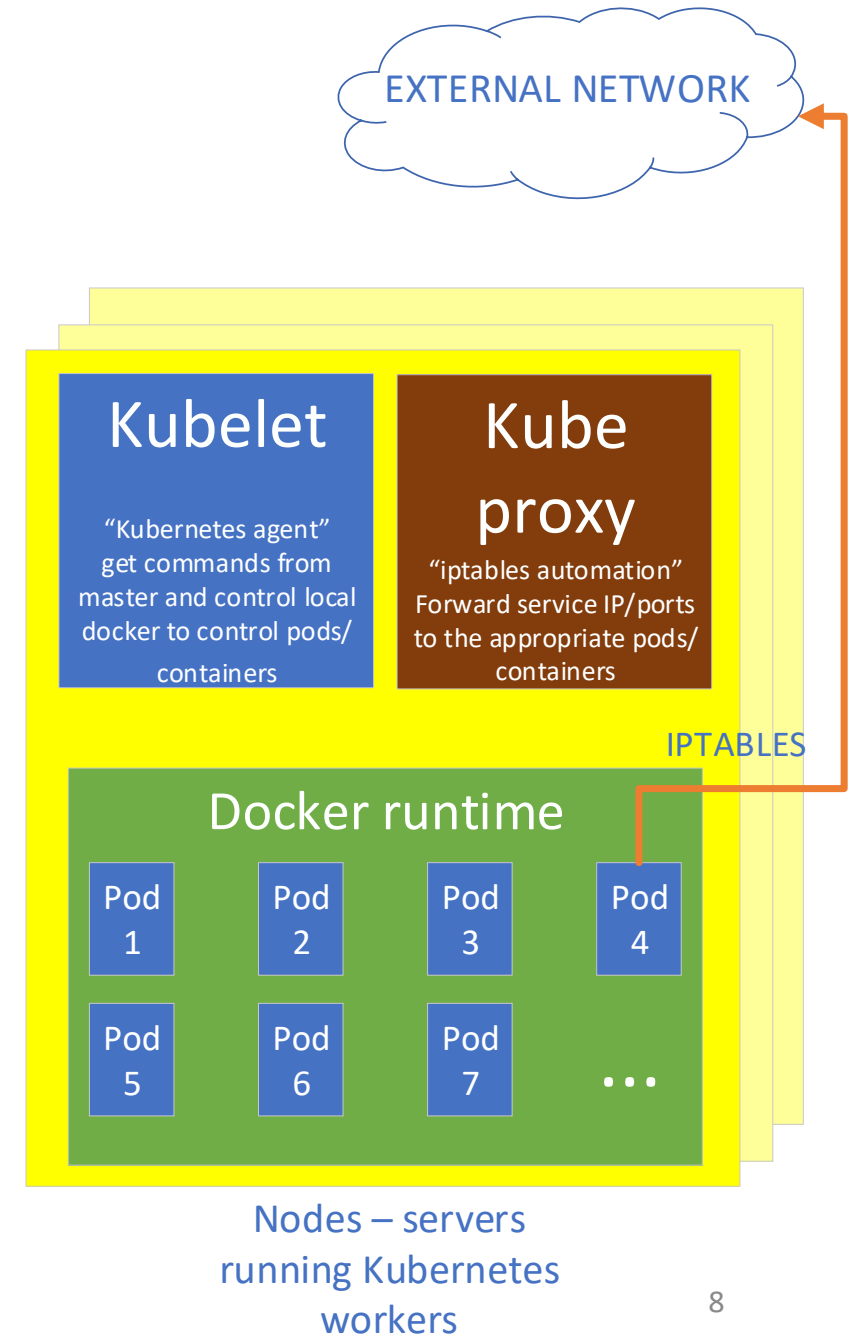
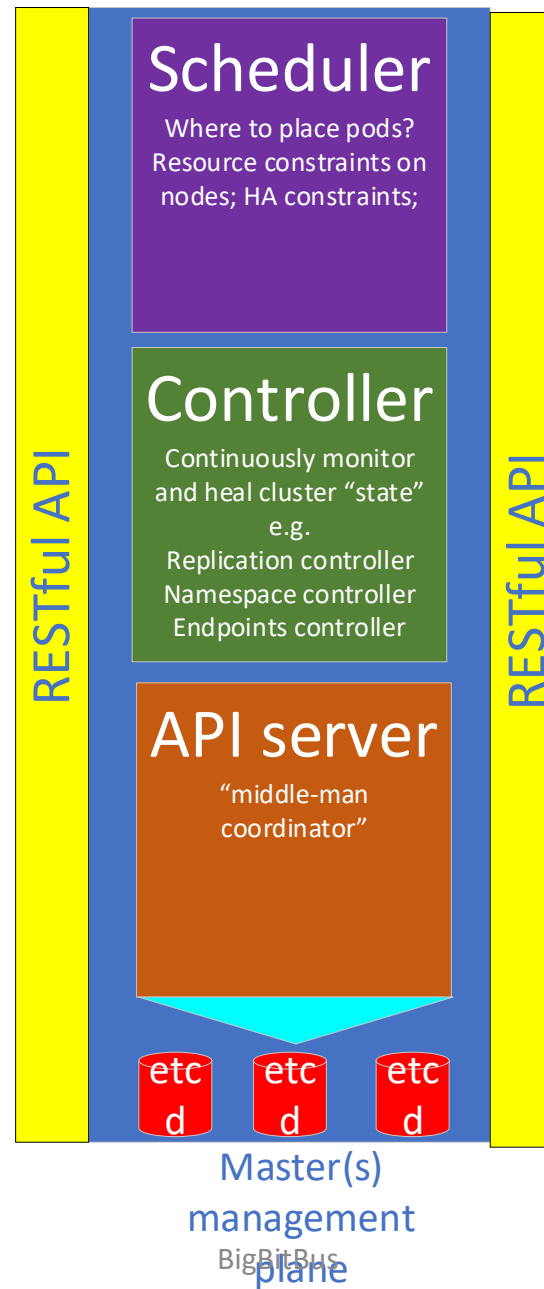


# Kubernetes Architecture

Multiple components

- Management plane
- Worker plane

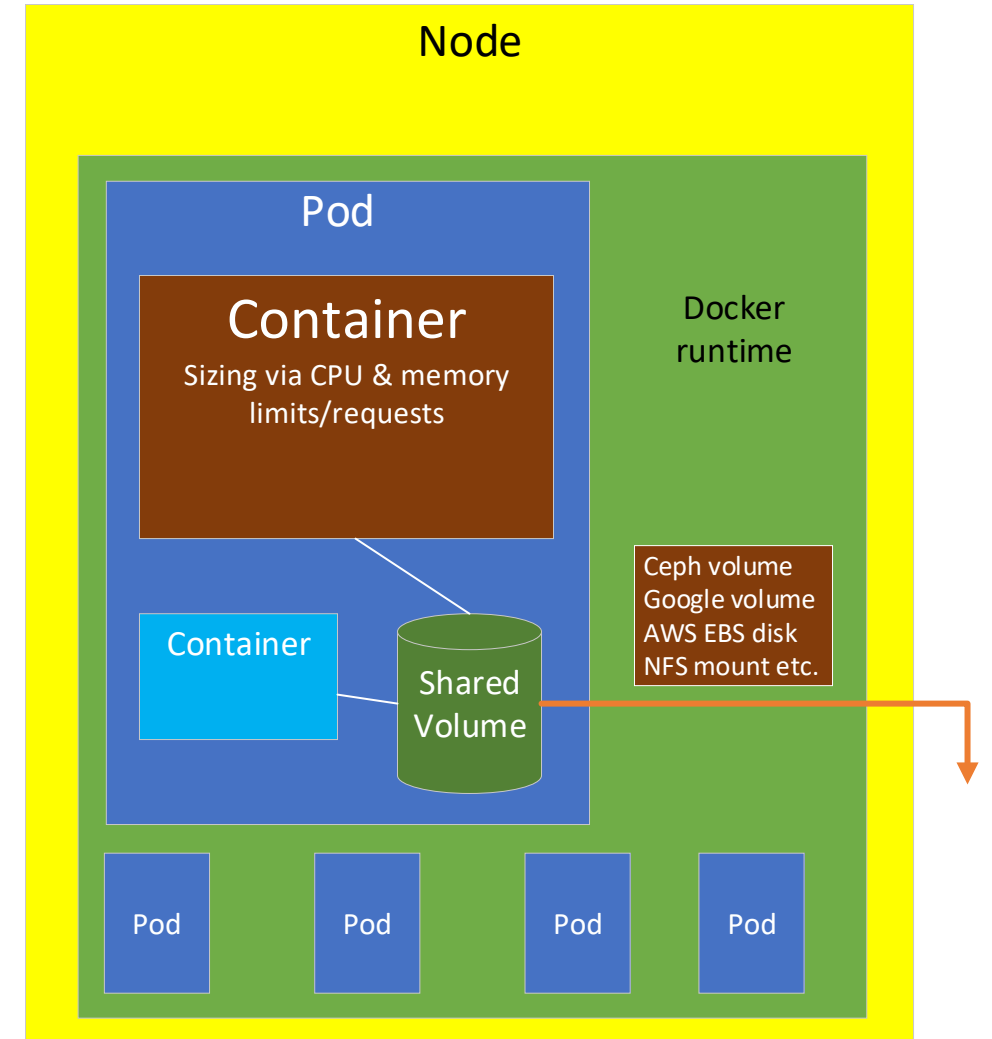
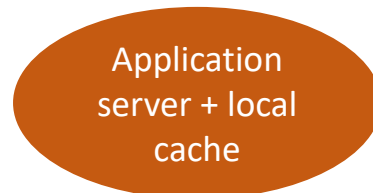
Worker nodes can be bare-metal or virtual machines





# Pod

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



# 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:/#
```

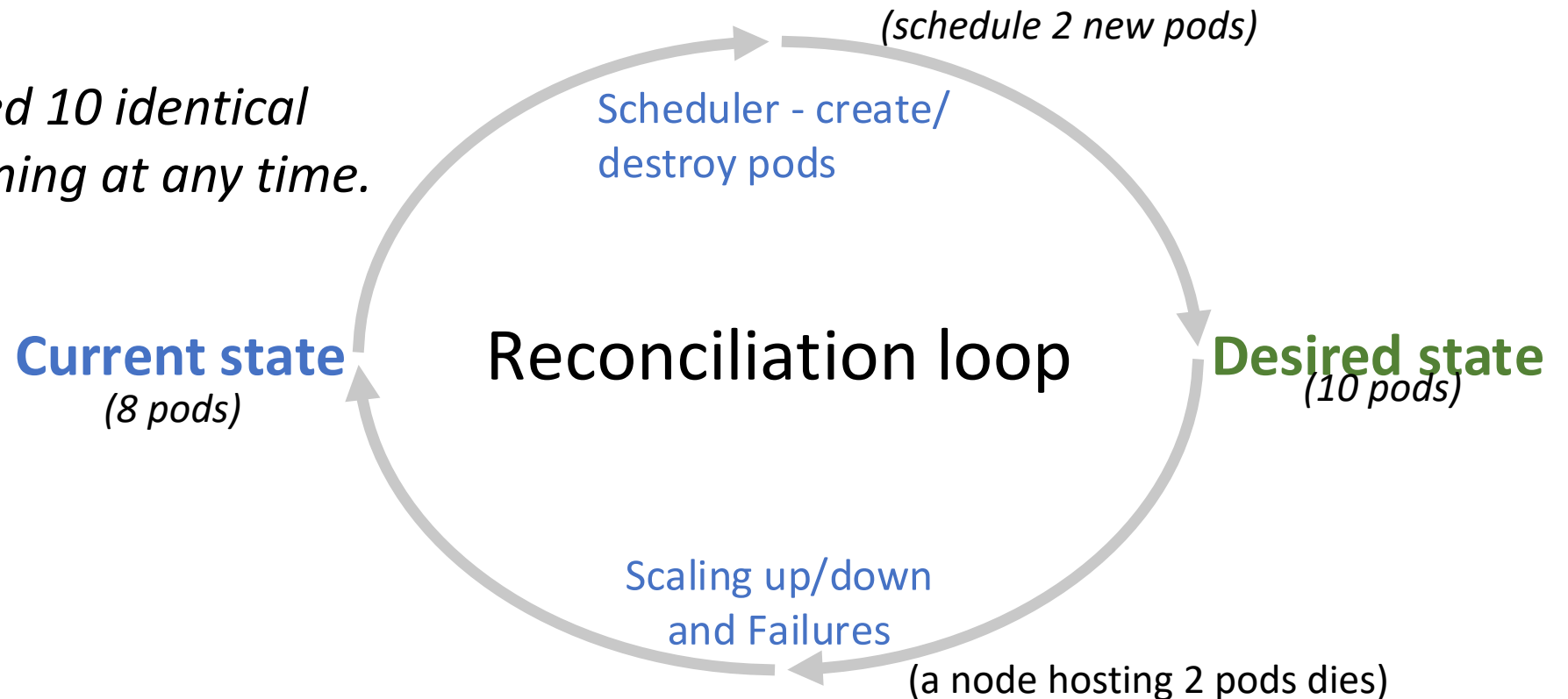


Running one-off pods is not the Kubernetes use-case

# Replicaset (of pods)

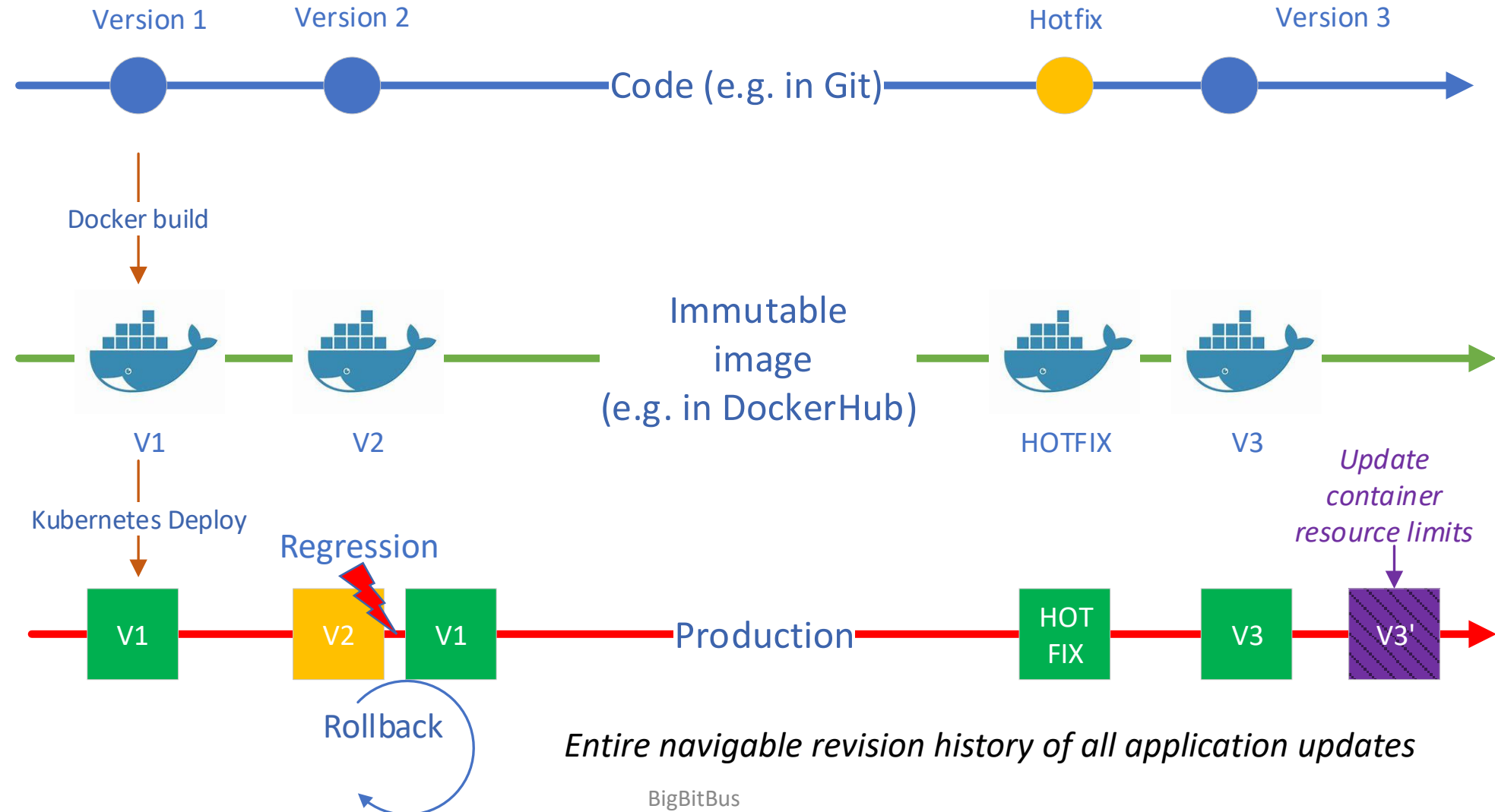
- Keep a defined number of identical pod replicas running in the Kubernetes cluster

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



Application updates, rollbacks, jumps to any revision

# Deployments



# 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



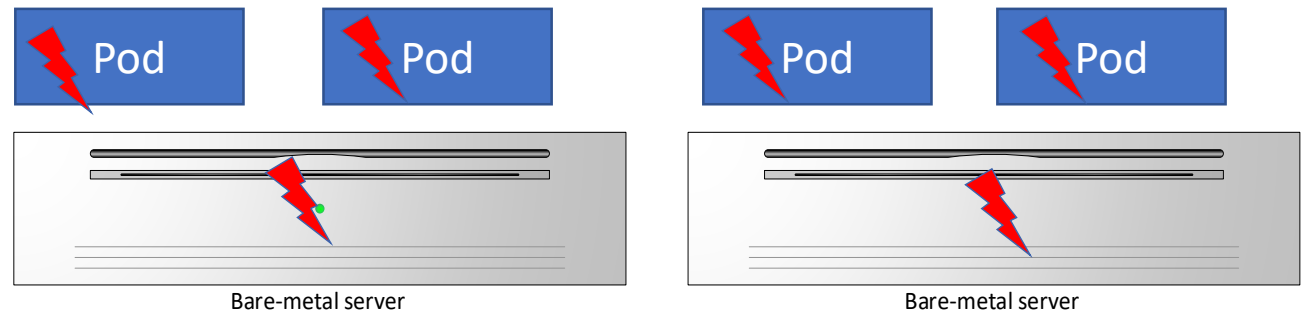
# Service Discovery

1. Assign a “cluster-IP” to a service to which other services within the cluster can connect. [Example](#)
2. Expose service externally (Load-balancer). [Example](#)
3. Connect to an external service. [Example](#)



*Pods are “cattle” - expendable, replaceable - not pets*

*Underlying pods and hardware are ephemeral and very dynamic*





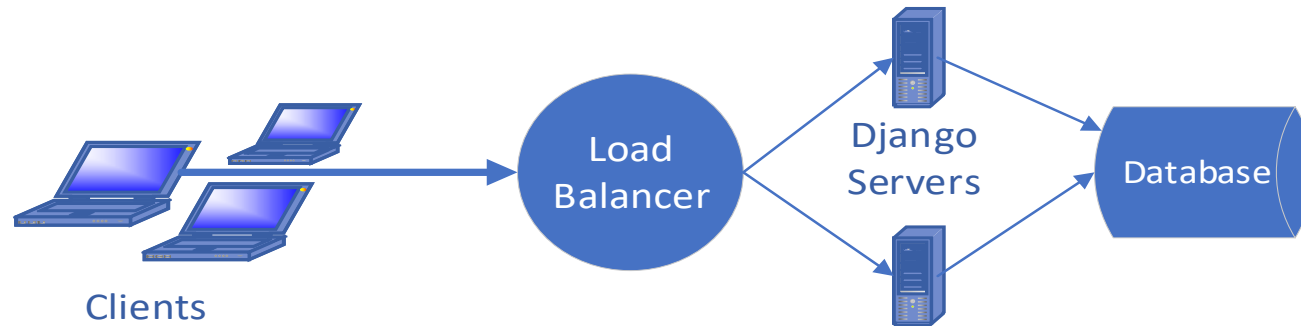
# Environment Variables & Secrets

- Inject meta-data into pods, e.g. based on namespace
- Configmaps for environment variables
  - Example ([development](#) vs. [production](#) namespace)
- Configmaps are very powerful e.g. for injecting custom scripts etc.
- Secrets
  - Create a secret

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

- [Expose secret](#) inside pod as an environment variable

# Polls Application Example



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

## Polling Application in Browser

### Polls

- [What is cool about Python?](#)
- [Which cloud provider is the best?](#)
- [Which star wars movie was the best?](#)

### Which cloud provider is the best?

- ☐ Digital Ocean
- ☐ Linode
- ☐ Google Cloud
- ☐ Amazon Web Services
- ☐ Microsoft Azure
- ☐ Other

Vote

## State definition - models.py

```
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

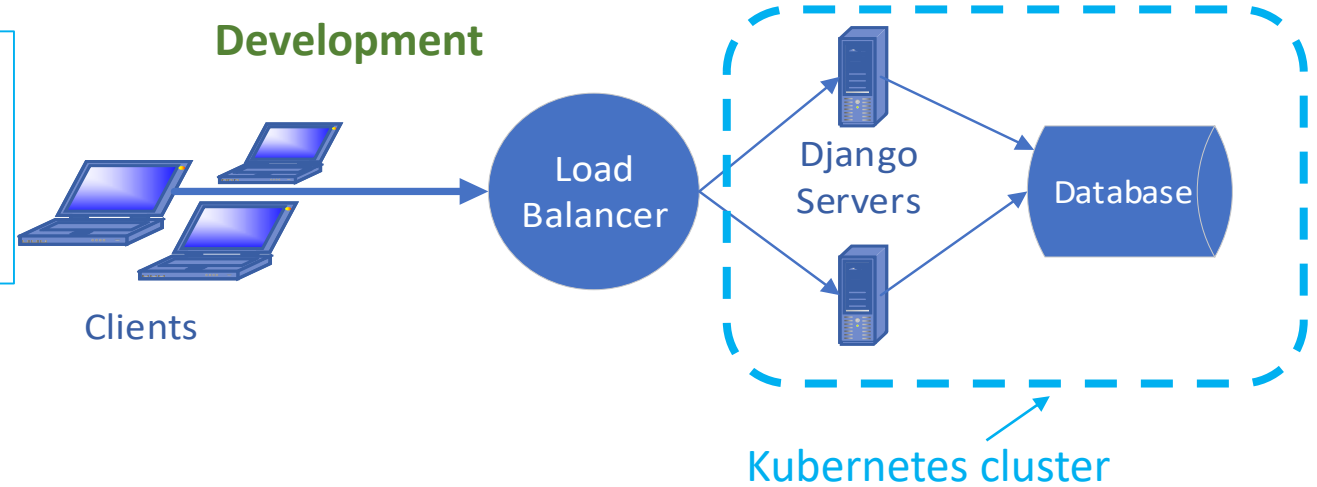
class Choice(models.Model):
    poll = models.ForeignKey(Poll, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

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

# Development and Production Setup

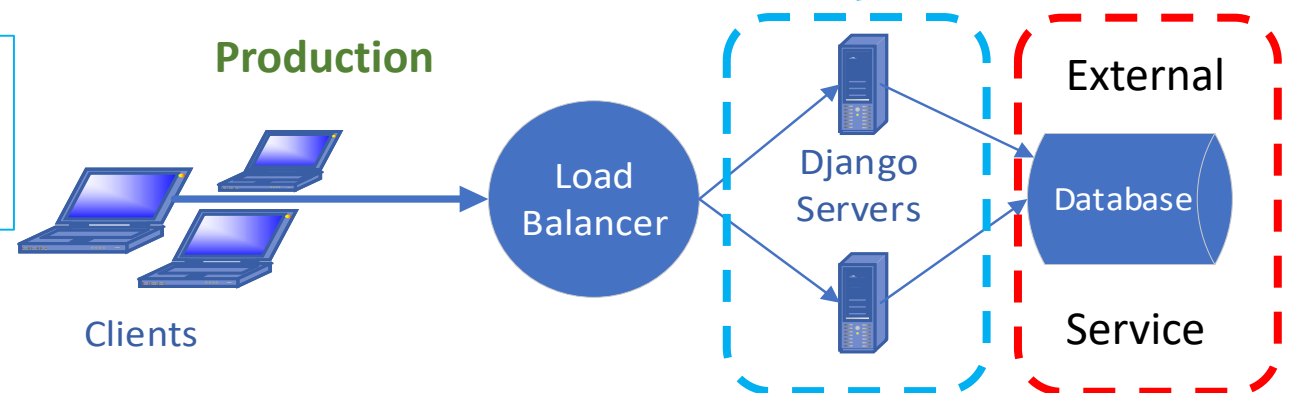
## Development

- Helps improve development and production code parity.



## Production

- Leave the stateful heavy-lifting to the experts.



[Django settings.py example](#)

[Developer workflow example](#)

# Other important concepts

- [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 replicaset): 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

# 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

# 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

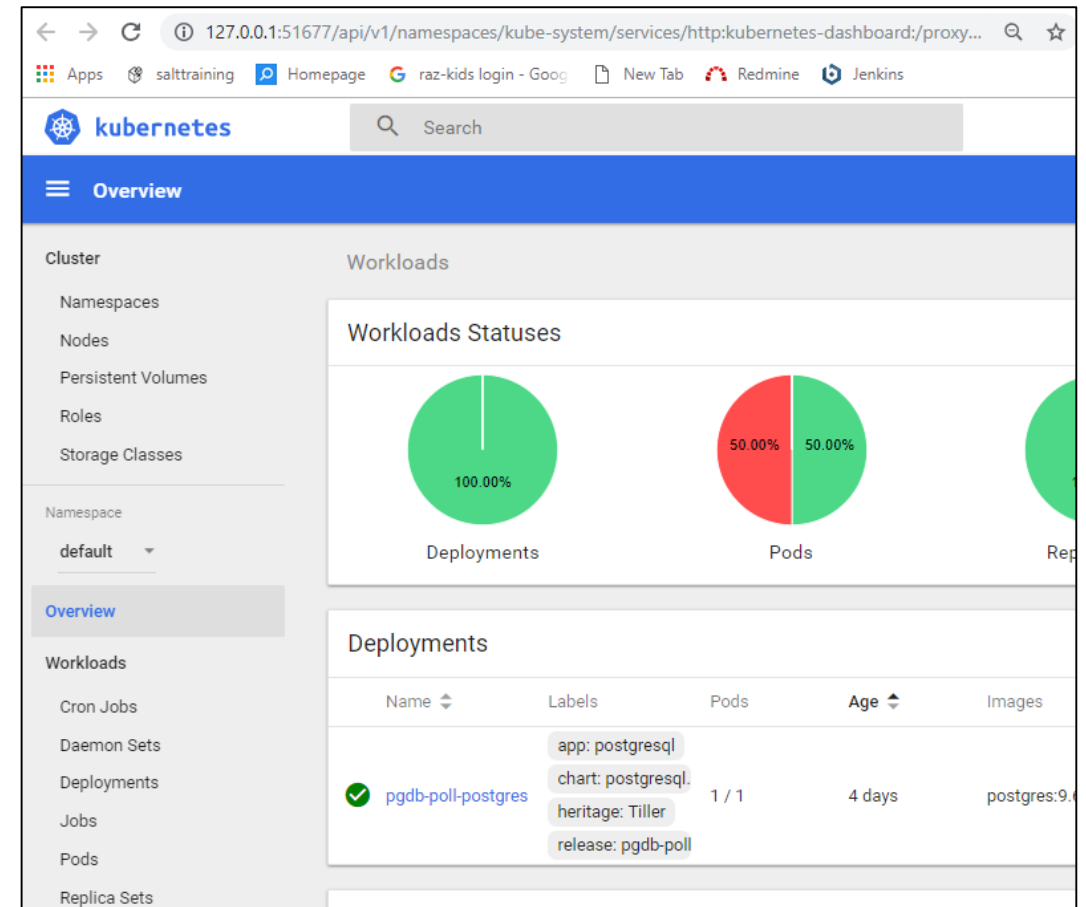
## minikube cli

```
# 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



# Beyond Manual Containers



## 1. Multiple node container runtime cluster

- Load-balancing across multiple containers
- Service discovery
- Fairness and resource sharing among different applications

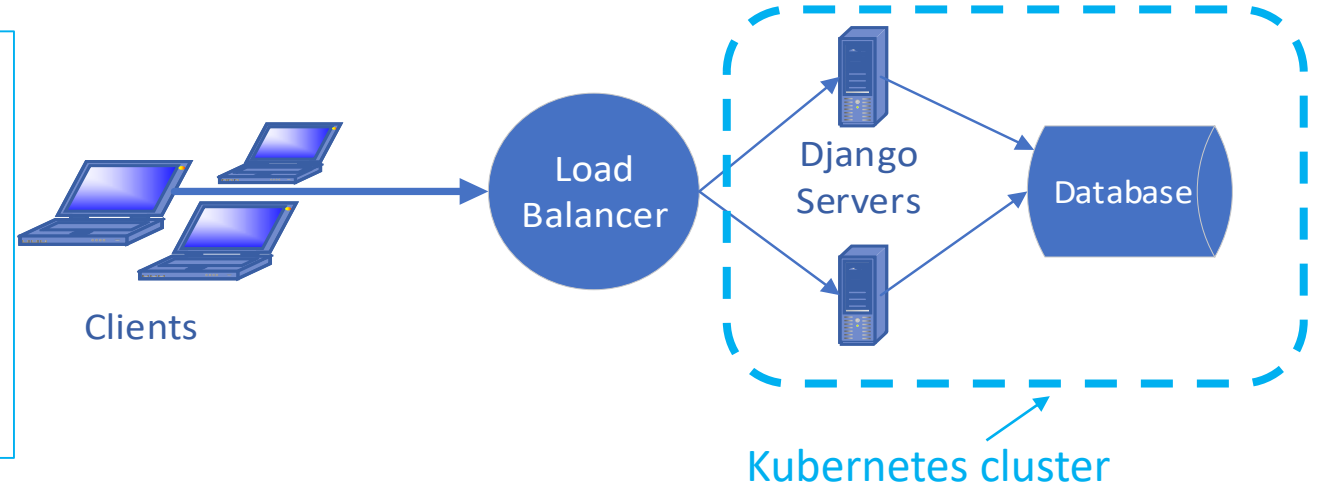
## 2. Production Operations Management

- Application updates & versioning
- Monitoring for failures; auto-healing
- Autonomous behavior – autoscaling and repairs
- Dealing with Failures
- Role based access controls
- Zero-downtime upgrades

# Stateful Databases in Kubernetes (?)

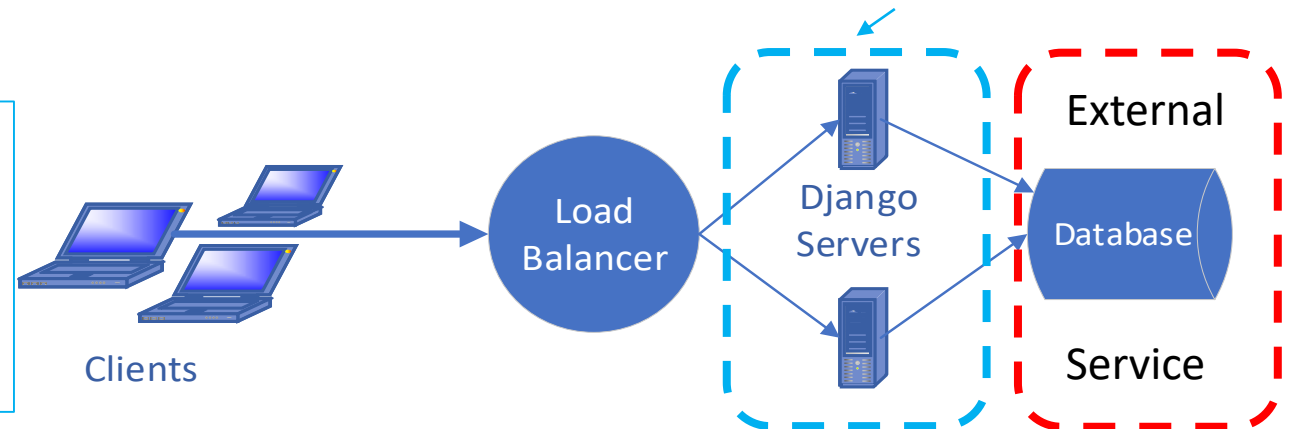
## Option 1

- 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



# BigBitBus

---

Transparency in public cloud and big data/analytics