AMRITA VISHWA VIDYAPEETHAM

Kubernetes

A way to build scalable and portable applications with Cloud

Dr Ganesh Neelakanta Iyer

Associate Professor, Dept of Computer Science and Engg

Amrita Vishwa Vidyapeetham, Coimbatore

The Need for Orchestration Systems

The Need for Orchestration Systems

- While Docker provided an open standard for packaging and distributing containerized applications, there arose a new problem
  - How would all of these containers be coordinated and scheduled?
  - How do all the different containers in your application communicate with each other?
  - How can container instances be scaled?

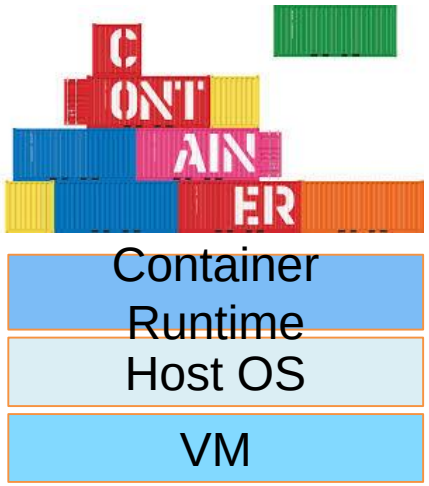
Solution
Container Orchestration Systems

kubernetes

MESOS

From Containers to Kubernetes

Container



Benefits

Isolation
Immutable infrastructure
Portability
Faster deployments
Versioning
Ease of sharing

Challenges

Networking
Deployments
Service Discovery
Auto Scaling
Persisting Data
Logging, Monitoring
Access Control

Container Scheduler


Kubernetes

Orchestration of cluster of containers across multiple hosts

- Automatic placements, networking, deployments, scaling, roll-out/-back, A/B testing

Declarative – not procedural

- Declare target state, reconcile to desired state
- Self-healing

Workload Portability

- Abstract from cloud provider specifics
- Multiple container runtimes

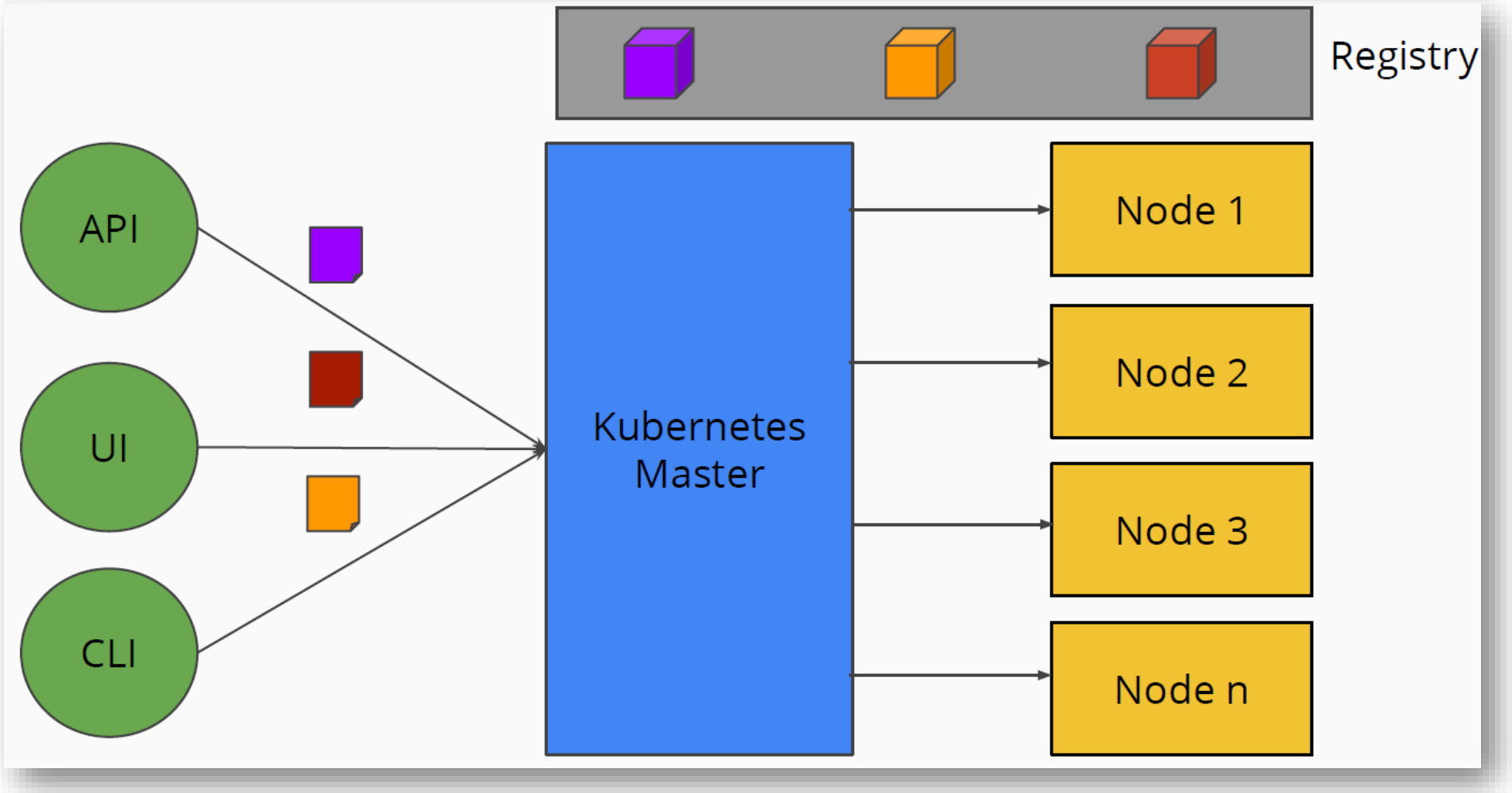
Kubernetes

- Kubernetes is an open-source container cluster manager
  - originally developed by Google, donated to the Cloud Native Computing Foundation
  - schedules & deploys containers onto a cluster of machines
    - e.g. ensure that a specified number of instances of an application are running
  - provides service discovery, distribution of configuration & secrets, ...
  - provides access to persistent storage
- Pod
  - smallest deployable unit of compute
  - consists of one or more containers that are always co-located, co-scheduled & run in a shared context

Why Kubernetes?

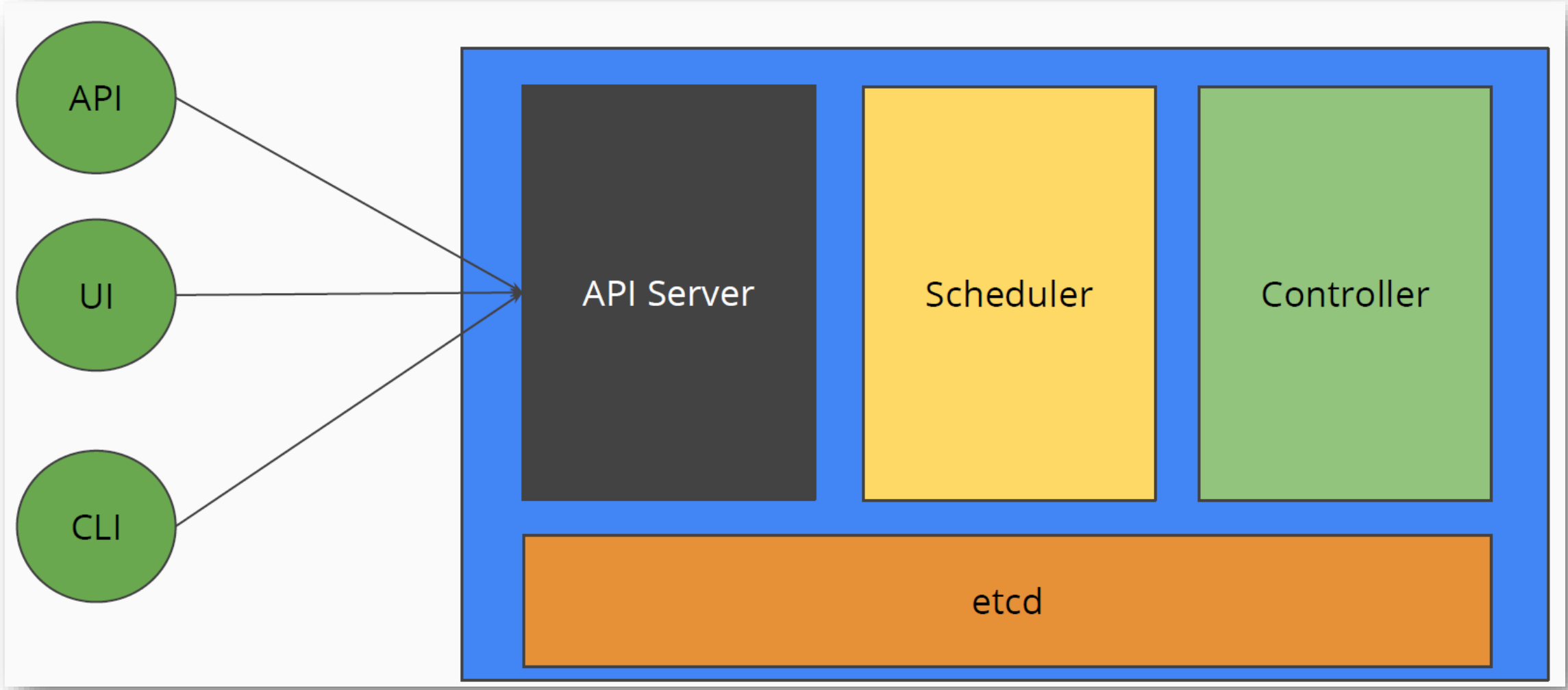
- It can be run anywhere
  - on-premises
    - bare metal, OpenStack, ...
  - public clouds
    - Google, Azure, AWS, ...
- Aim is to use Kubernetes as an abstraction layer
  - migrate to containerised applications managed by Kubernetes & use only the Kubernetes API
  - can then run out-of-the-box on any Kubernetes cluster
- Avoid vendor lock-in as much as possible by not using any vendor specific APIs or services
  - except where Kubernetes provides an abstraction
    - e.g. storage, load balancers

Kubernetes Architecture



https://www.slideshare.net/janakiramm/kubernetes-architecture

Kubernetes Master



https://www.slideshare.net/janakiramm/kubernetes-architecture

kube-apiserver

- The apiserver provides a forward facing REST interface into the kubernetes control plane and datastore
- All clients, including nodes, users and other applications interact with kubernetes strictly through the API Server
- It is the true core of Kubernetes acting as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore

etcd

- Etcd acts as the cluster datastore
- Providing a strong, consistent and highly available key-value store used for persisting cluster state

kube-controller-manager

- The controller-manager is the primary daemon that manages all core component control loops
- It monitors the cluster state via the apiserver and steers the cluster towards the desired state
- These controllers include:
  - Node Controller: Responsible for noticing and responding when nodes go down.
  - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
  - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces

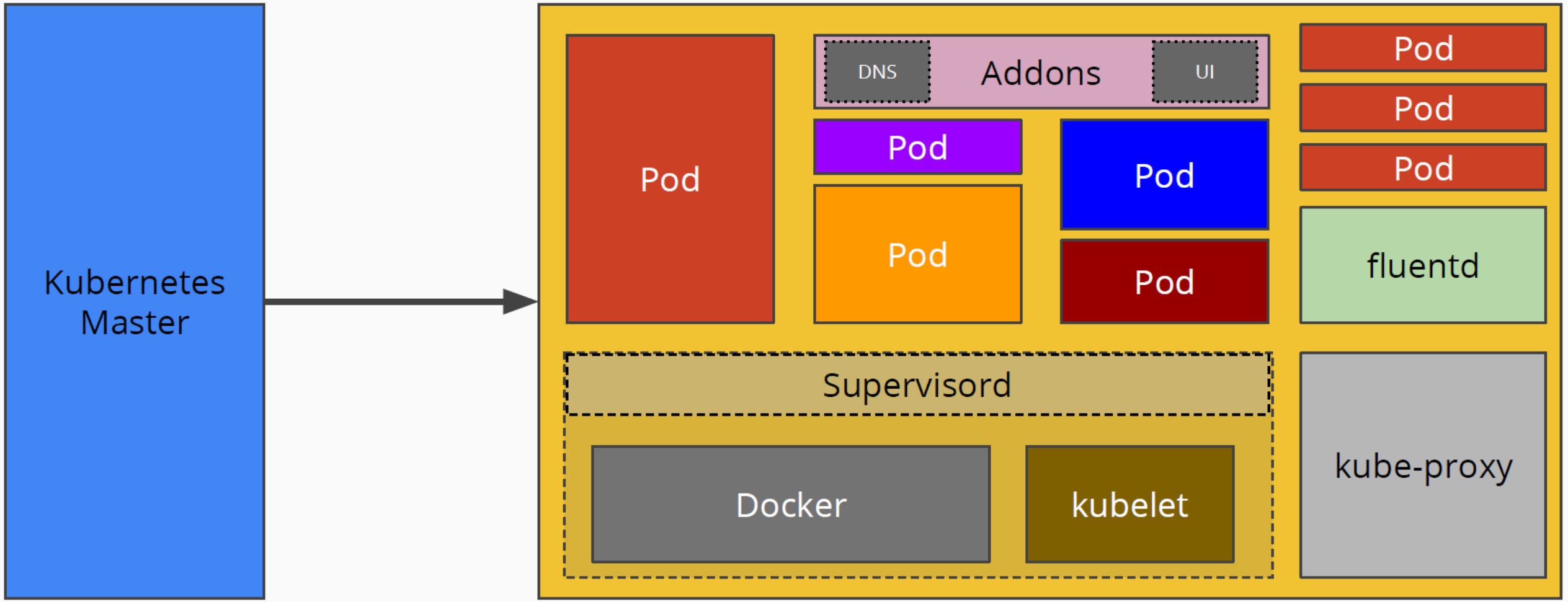
cloud-controller-manager

- cloud-controller-manager runs controllers that interact with the underlying cloud providers
- cloud-controller-manager allows cloud vendors code and the Kubernetes code to evolve independent of each other

kube-scheduler

- Kube-scheduler is a verbose policy-rich engine that evaluates workload requirements and attempts to place it on a matching resource
- These requirements can include such things as general hardware reqs, affinity, anti-affinity, and other custom resource requirements

Kubernetes Node



https://www.slideshare.net/janakiramm/kubernetes-architecture

Pod

- A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy
- A Pod represents a running process on your cluster
- A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run
- A Pod represents a unit of deployment: *a single instance of an application in Kubernetes*, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources

kubelet

- An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn’t manage containers which were not created by Kubernetes

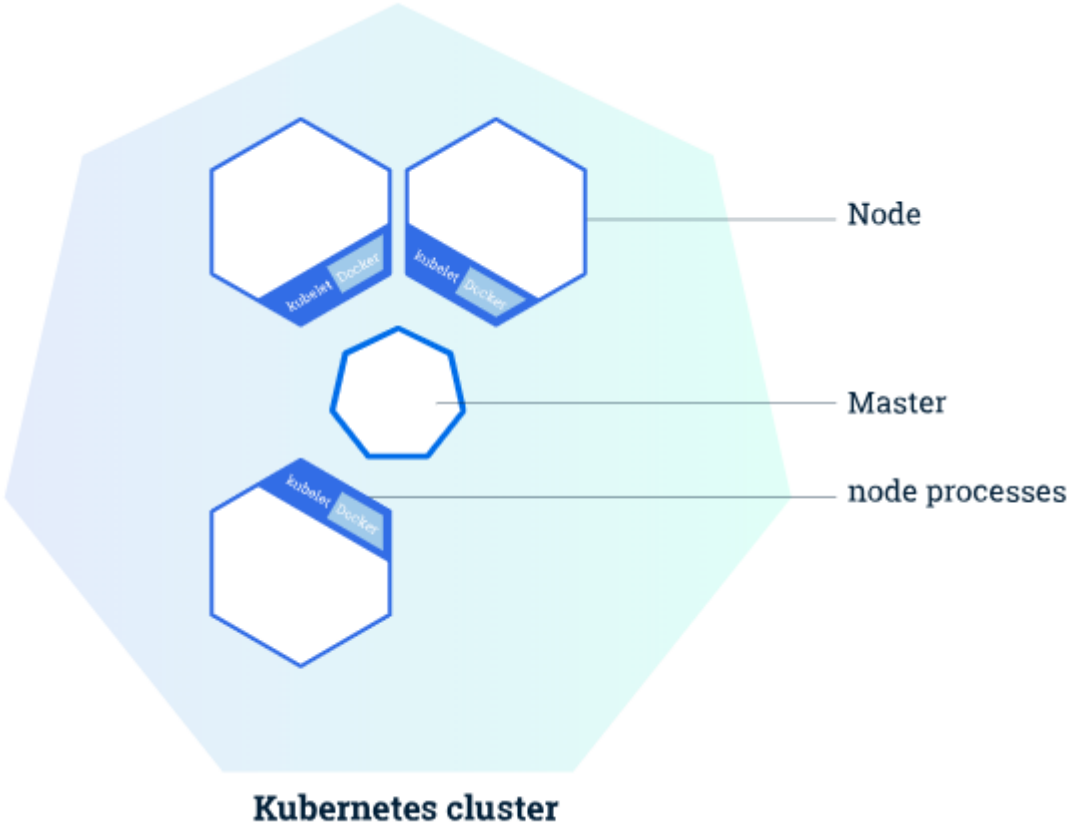
kube-proxy

- Enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding

Container Runtime

- The container runtime is the software that is responsible for running containers
- Kubernetes supports several runtimes
  - Docker, rkt, runc and any OCI runtime-spec implementation

Kubernetes Cluster



- Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit
- Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way

Running Kubernetes Locally via Minikube

- Minikube is a tool that makes it easy to run Kubernetes locally
- Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day


BOSTON • DUCK • TOURS
LAND & WATER
Hello Minikube

Hello Minicube

- This tutorial provides a container image built from the following files

minikube/server.js

```
var http = require('http');

var handleRequest = function(request, response) {
  console.log('Received request for URL: ' + request.url);
  response.writeHead(200);
  response.end('Hello World!');
};
var www = http.createServer(handleRequest);
www.listen(8080);
```

minikube/Dockerfile

```
FROM node:6.14.2
EXPOSE 8080
COPY server.js .
CMD node server.js
```

Create a minikube cluster

- minikube version
- minikube start
- minikube dashboard

Create a Deployment

- A Kubernetes Pod is a group of one or more Containers, tied together for the purposes of administration and networking
- The Pod in this tutorial has only one Container
- A Kubernetes Deployment checks on the health of your Pod and restarts the Pod's Container if it terminates
- Deployments are the recommended way to manage the creation and scaling of Pods

Create a Deployment

- Use the `kubectl create` command to create a Deployment that manages a Pod
- The Pod runs a Container based on the provided Docker image

```
kubectl create deployment hello-node --image=
gcr.io/hello-minikube-zero-install/hello-node
```

```
PS C:\Users\GiRi> kubectl create deployment hello-node --image=gcr.io/hello-minikube-zero-
install/hello-node
deployment.apps/hello-node created
```

Create a Deployment

View the deployment

```
kubectl get deployments
```

```
PS C:\Users\GiRi> kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
hello-node   1/1     1            1           3m3s
PS C:\Users\GiRi>
```

Create a Deployment

- View the Pod

```
kubectl get pods
```

```
PS C:\Users\GiRi> kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-node-64c578bdf8-s5lxx   1/1     Running   0          5m13s
PS C:\Users\GiRi>
```

Create a deployment

- View cluster events

```
kubectl get events
```

- View the kubectl configuration

```
kubectl config view
```

Create s Service

- By default, the Pod is only accessible by its internal IP address within the Kubernetes cluster
- To make the hello-node Container accessible from outside the Kubernetes virtual network, you have to expose the Pod as a Kubernetes Service
- Expose the Pod to the public internet using the kubectl expose command

```
kubectl expose deployment hello-node --type=LoadBalancer --port=8080
```

Create a Service

- View the Service you just created

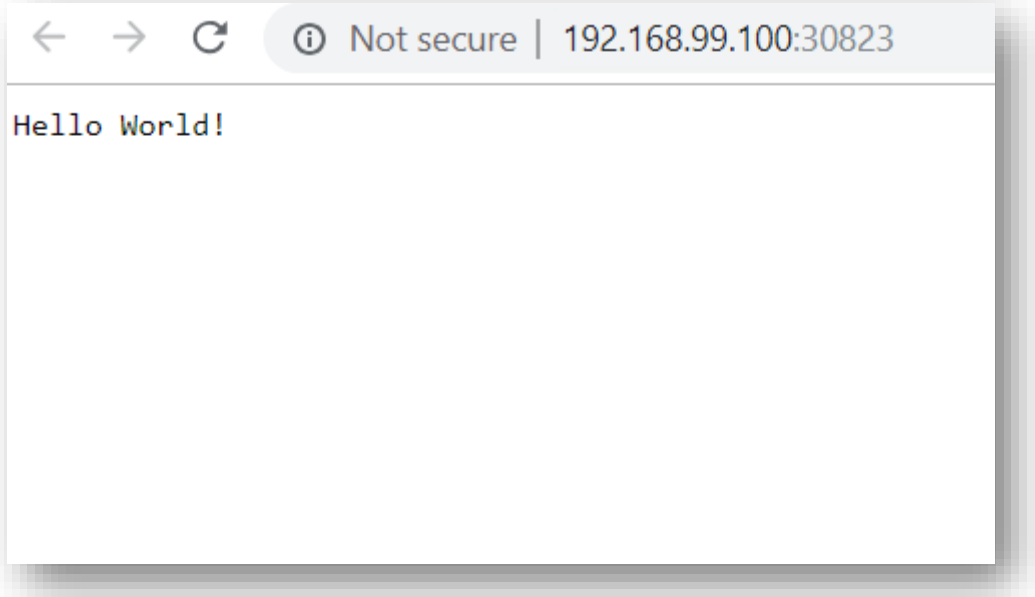
```
kubectl get services
```

```
PS C:\Users\GiRi> kubectl get services
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-node   LoadBalancer   10.102.190.217   <pending>     8080:30823/TCP   80s
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP          4d9h
```

Run a Service

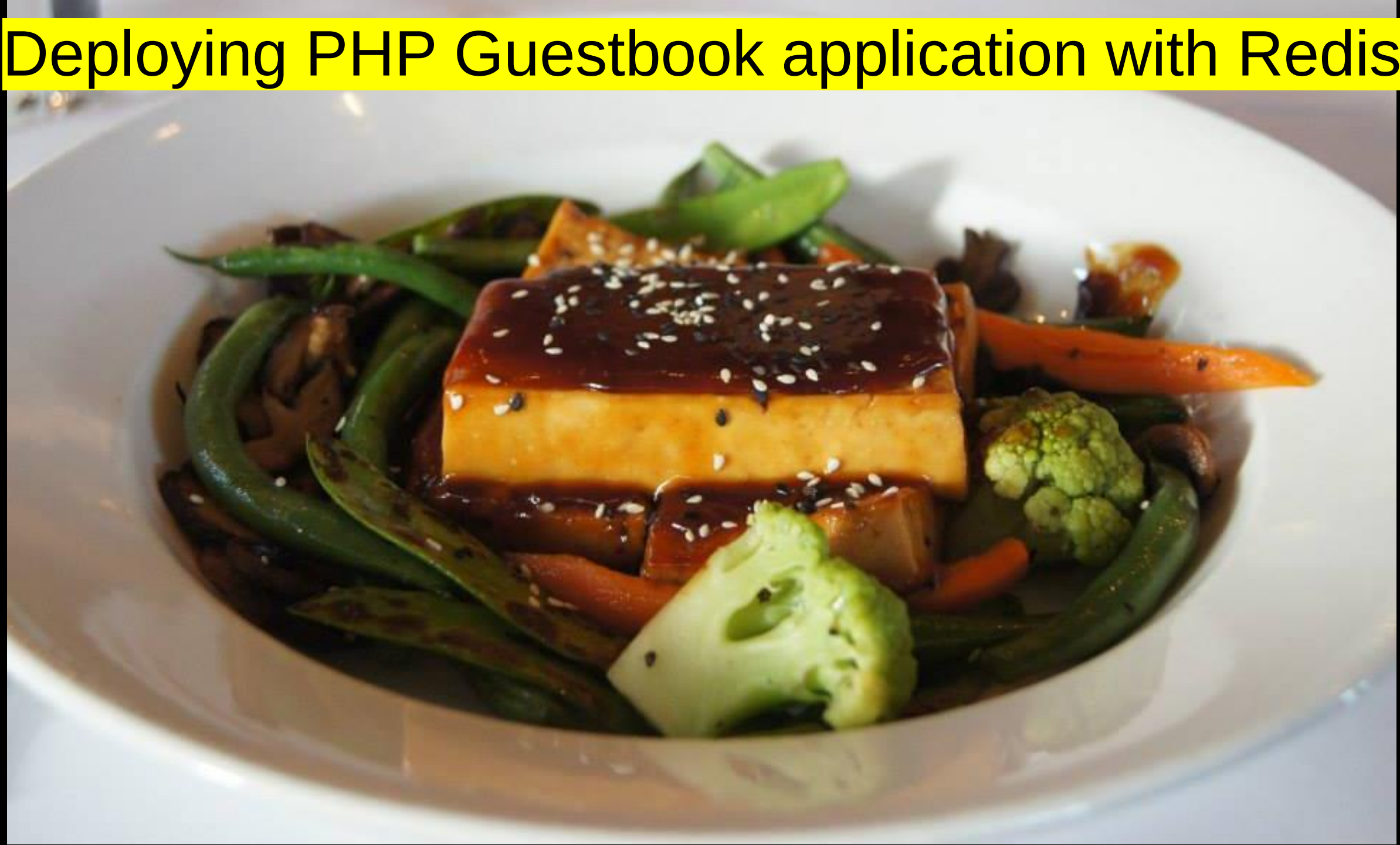
- Run the following command

```
minikube service hello-node
```



Bigger Experiment with Kubernetes

Deploying PHP Guestbook application with Redis



Deploying PHP Guestbook application with Redis

- This tutorial shows you how to build and deploy a simple, multi-tier web application using Kubernetes and Docker
- This example consists of the following components:
  - A single-instance Redis master to store guestbook entries
  - Multiple replicated Redis instances to serve reads
  - Multiple web frontend instances

Objectives

- Start up a Redis master
- Start up Redis slaves
- Start up the guestbook frontend
- Expose and view the Frontend Service

Start up the Redis Master

- The guestbook application uses Redis to store its data
- It writes its data to a Redis master instance and reads data from multiple Redis slave instances
- Creating the Redis Master Deployment
- Copy the folder here to your system

https://tinyurl.com/anokadockers

*.yaml file

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis:e2e  # or just image: redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

Start up the Redis Master

- Launch a terminal window in the directory you downloaded the manifest files
- Apply the Redis Master Deployment from the redis-master-deployment.yaml file

```
kubectl apply -f redis-master-deployment.yaml
```

```
$ kubectl apply -f redis-master-deployment.yaml
deployment.extensions/redis-master created
```

Start up the Redis Master

- Query the list of Pods to verify that the Redis Master Pod is running:

```
kubectl get pods
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/Exter
nal/Kubernetes/application/guestbook
$  kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-node-64c578bdf8-s5lxx   1/1     Running   0          24m
redis-master-6fbbc44567-bqkw8   1/1     Running   0          5m46s
```

Run the following command to view the logs from the Redis Master Pod

```
kubectl logs -f POD-NAME
```

Replace POD-NAME with the name of your Pod

```
$  kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-node-64c578bdf8-s5lxx    1/1     Running   0          24m
redis-master-6fbbc44567-bqkw8  1/1     Running   0          5m46s

GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestboo
k
$  kubectl logs -f POD-NAME
Error from server (NotFound): pods "POD-NAME" not found

GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestboo
k
$  kubectl logs -f redis-master-6fbbc44567-bqkw8
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 2.8.19 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Running in stand alone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
      |    `-._   `._    /     _.-'    |     PID: 1
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |           http://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'

[1] 15 Feb 17:05:26.250 # Server started, Redis version 2.8.19
[1] 15 Feb 17:05:26.250 # WARNING: The TCP backlog setting of 511 cannot be enforced becau
se /proc/sys/net/core/somaxconn is set to the lower value of 128.
[1] 15 Feb 17:05:26.250 * The server is now ready to accept connections on port 6379
```

Creating the Redis Master Service

- The guestbook applications needs to communicate to the Redis master to write its data
- You need to apply a Service to proxy the traffic to the Redis master Pod
- A Service defines a policy to access the Pods
- Launch a terminal window in the directory you downloaded the manifest files
- Apply the Redis Master Service from the following redis-master-service.yaml file

```
kubectl apply -f redis-master-service.yaml
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/applicatio
n/guestbook
$ kubectl apply -f redis-master-service.yaml
service/redis-master created
```

Creating the Redis Master Service

- Query the list of Services to verify that the Redis Master Service is running
- `kubectl get service`

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/applicatio
n/guestbook
$ kubectl get service
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-node     LoadBalancer   10.102.190.217   <pending>     8080:30823/TCP   24m
kubernetes     ClusterIP      10.96.0.1        <none>        443/TCP          4d10h
redis-master   ClusterIP      10.96.190.134    <none>        6379/TCP         109s
```

Start up the Redis Slaves

- Although the Redis master is a single pod, you can make it highly available to meet traffic demands by adding replica Redis slaves

Creating the Redis Slave Deployment

- Deployments scale based off of the configurations set in the manifest file. In this case, the Deployment object specifies two replicas
- If there are not any replicas running, this Deployment would start the two replicas on your container cluster
- Conversely, if there are more than two replicas are running, it would scale down until two replicas are running

Creating the Redis Slave Deployment

- Apply the Redis Slave Deployment from the redis-slave-deployment.yaml file

```
kubectl apply -f redis-slave-deployment.yaml
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl apply -f redis-slave-deployment.yaml
deployment.extensions/redis-slave created

GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$
```

Creating the Redis Slave Deployment

- Query the list of Pods to verify that the Redis Slave Pods are running:

```
kubectl get pods
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-node-64c578bdf8-s5lxx    1/1     Running   0          38m
redis-master-6fbbc44567-bqkw8  1/1     Running   0          18m
redis-slave-74ccb764fc-5sjcm   1/1     Running   0          69s
redis-slave-74ccb764fc-krnwz   1/1     Running   0          69s
```

Creating the Redis Slave Service

- The guestbook application needs to communicate to Redis slaves to read data
- To make the Redis slaves discoverable, you need to set up a Service
- A Service provides transparent load balancing to a set of Pods

Creating the Redis Slave Service

- Apply the Redis Slave Service from the following redis-slave-service.yaml file

```
kubectl apply -f redis-slave-service.yaml
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl apply -f redis-slave-service.yaml
service/redis-slave created
```

Creating the Redis Slave Service

- Query the list of Services to verify that the Redis slave service is running

```
kubectl get services
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl get services
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-node     LoadBalancer   10.102.190.217   <pending>     8080:30823/TCP   33m
kubernetes     ClusterIP      10.96.0.1        <none>        443/TCP          4d10h
redis-master   ClusterIP      10.96.190.134    <none>        6379/TCP         11m
redis-slave    ClusterIP      10.106.182.175   <none>        6379/TCP         93s
```

Set up and Expose the Guestbook Frontend

- The guestbook application has a web frontend serving the HTTP requests written in PHP
- It is configured to connect to the redis-master Service for write requests and the redis-slave service for Read requests

Creating the Guestbook Frontend Deployment

- Apply the frontend Deployment from the frontend-deployment.yaml file

```
kubectl apply -f frontend-deployment.yaml
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl apply -f frontend-deployment.yaml
deployment.extensions/frontend created
```

Creating the Guestbook Frontend Deployment

- Query the list of Pods to verify that the three frontend replicas are running

```
kubectl get pods -l app=guestbook -l
tier=frontend
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl get pods -l app=guestbook -l tier=frontend
NAME                        READY   STATUS    RESTARTS   AGE
frontend-74b4665db5-98ppg   1/1     Running   0          66s
frontend-74b4665db5-ht7zk   1/1     Running   0          66s
frontend-74b4665db5-mx2mf   1/1     Running   0          66s
```

Creating the frontend service

- The redis-slave and redis-master Services you applied are only accessible within the container cluster because the default type for a Service is ClusterIP
- ClusterIP provides a single IP address for the set of Pods the Service is pointing to
- This IP address is accessible only within the cluster.
- If you want guests to be able to access your guestbook, you must configure the frontend Service to be externally visible, so a client can request the Service from outside the container cluster
- Minikube can only expose Services through NodePort

Creating the frontend service

- Apply the frontend Service from the frontend-service.yaml file

```
kubectl apply -f frontend-service.yaml
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl apply -f frontend-service.yaml
service/frontend created
```

Creating the frontend service

- Query the list of Services to verify that the frontend Service is running

```
kubectl get services
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl get services
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
frontend       NodePort       10.107.134.150   <none>        80:32666/TCP     55s
hello-node     LoadBalancer   10.102.190.217   <pending>     8080:30823/TCP   41m
kubernetes     ClusterIP      10.96.0.1        <none>        443/TCP          4d10h
redis-master   ClusterIP      10.96.190.134    <none>        6379/TCP         19m
redis-slave    ClusterIP      10.106.182.175   <none>        6379/TCP         9m46s
```

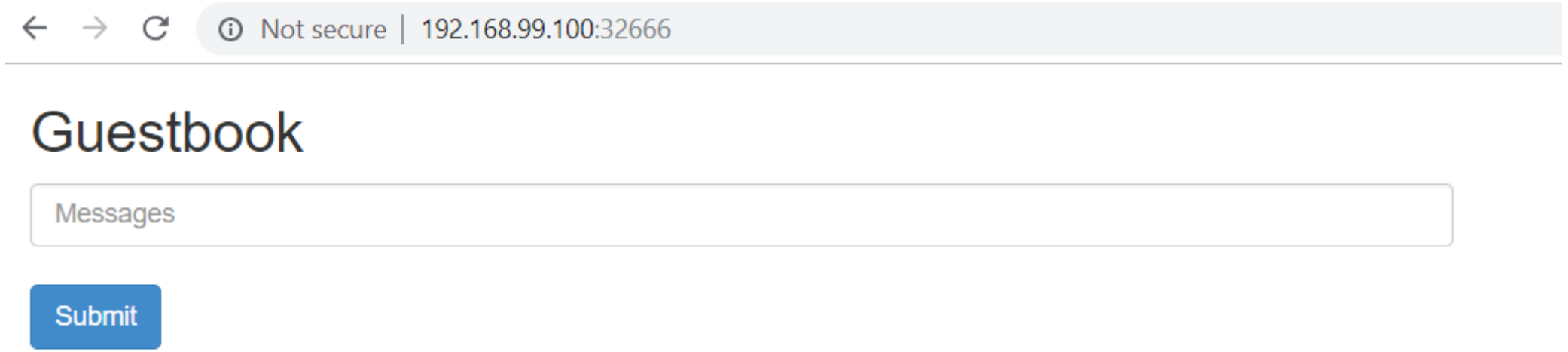
Viewing the Frontend Service via NodePort

- If you deployed this application to Minikube or a local cluster, you need to find the IP address to view your Guestbook
- Run the following command to get the IP address for the frontend Service

```
minikube service frontend --url
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ minikube service frontend --url
http://192.168.99.100:32666
```

Go to a browser and type that URL



Viewing the Frontend Service via LoadBalancer

- If you deployed the frontend-service.yaml manifest with type: LoadBalancer you need to find the IP address to view your Guestbook
- Run the following command to get the IP address for the frontend Service

```
kubectl get service frontend
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kubernetes/application/guestbook
$ kubectl get service frontend
NAME       TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
frontend   NodePort   10.107.134.150   <none>        80:32666/TCP   5m11s
```

Scale the Web Frontend

- Scaling up or down is easy because your servers are defined as a Service that uses a Deployment controller
- Run the following command to scale up the number of frontend Pods:

```
kubectl scale deployment frontend --replicas=5
```

- Query the list of Pods to verify the number of frontend Pods running:

```
kubectl get pods
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/External/Kuberne
$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
frontend-74b4665db5-72rp2         1/1     Running   0          7s
frontend-74b4665db5-98ppg         1/1     Running   0          11m
frontend-74b4665db5-ht7zk         1/1     Running   0          11m
frontend-74b4665db5-mx2mf         1/1     Running   0          11m
frontend-74b4665db5-w7lrw         1/1     Running   0          7s
hello-node-64c578bdf8-s5lxx       1/1     Running   0          57m
redis-master-6fbbc44567-bqkw8     1/1     Running   0          38m
redis-slave-74ccb764fc-5sjcm      1/1     Running   0          20m
redis-slave-74ccb764fc-krnwz      1/1     Running   0          20m
```

Summary

- Kubernetes can help you
  - Create clusters
  - Deploy applications
  - Scale your business


adform

AMADEUS

ancestry

Bla Bla Car

BLACKROCK

box

buffer

CCP

CapitalOne

COMCAST

CONCUR.

Crowdfire

ebay

Goldman
Sachs

GOLFNOW

HAUFE.Group

Home Office

HUAWEI

IBM

ING

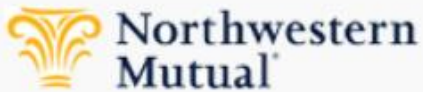

JD.京东
.COM

LIVEPERSON

monzo

NAIC
National Association of Insurance Commissioners

The New York Times

NORDSTROM
TECHNOLOGY

Northwestern Mutual

ocado

OpenAI

Pear Deck

Pearson

PHILIPS

Pinterest

POKÉMON
GO

SAP

SAMSUNG
SAMSUNG SDS

sling
TELEVISION

SOUNDCLOUD

SQUARESPACE

wepay

Thank you!

Dr Ganesh Neelakanta Iyer

ni_amrita@cb.amrita.edu
ganesh.vigneswara@gmail.com

GANESHNIYER

श्रद्धावान् लभते ज्ञानम्