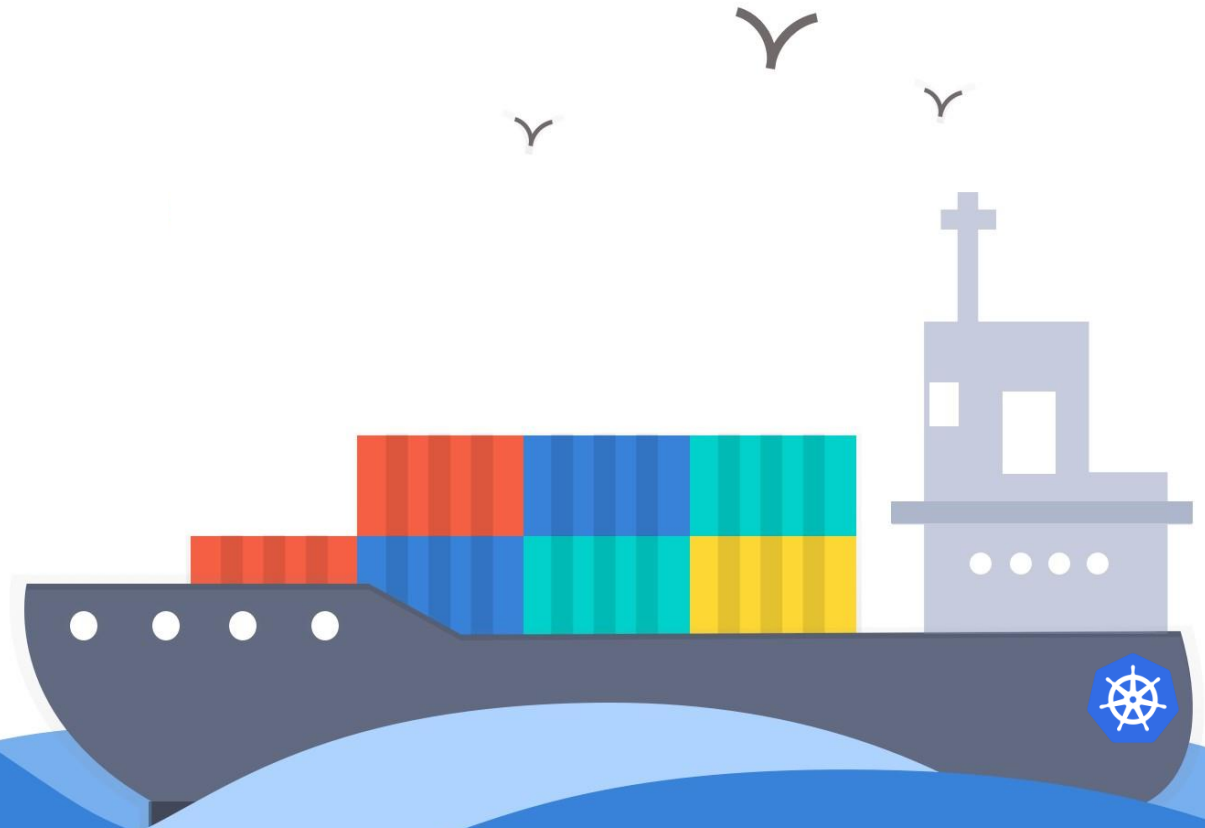


Getting Started with **K3S**



Contents



K3S



- K3s Architecture
- Application Deployments
- Services
- Docker Images for ARM32 architecture
- Configuring Ingress Rules
- Accessing Kubernetes & Traefik dashboards





Kubernetes

- Kubernetes also known as **K8s**, is an open-source Container Management tool
- It provides a container runtime, container orchestration, container-centric infrastructure orchestration, self-healing mechanisms, service discovery, load balancing and container (de)scaling
- Initially developed by Google, for managing containerized applications in a clustered environment but later donated to **CNCF**
- Written in **Golang**
- It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability





| Kubernetes

Certified Kubernetes Distributions

- Cloud Managed: **EKS** by AWS, **AKS** by Microsoft and **GKE** by google
- Self Managed: **OpenShift** by Redhat and **Docker Enterprise**
- Local dev/test: **Micro K8s** by Canonical, **Minikube**
- Vanilla Kubernetes: The core Kubernetes project(baremetal), **Kubeadm**
- Special builds: **K3s** by Rancher, a light weight K8s distribution for Edge devices

Online Emulator: <https://labs.play-with-k8s.com/>



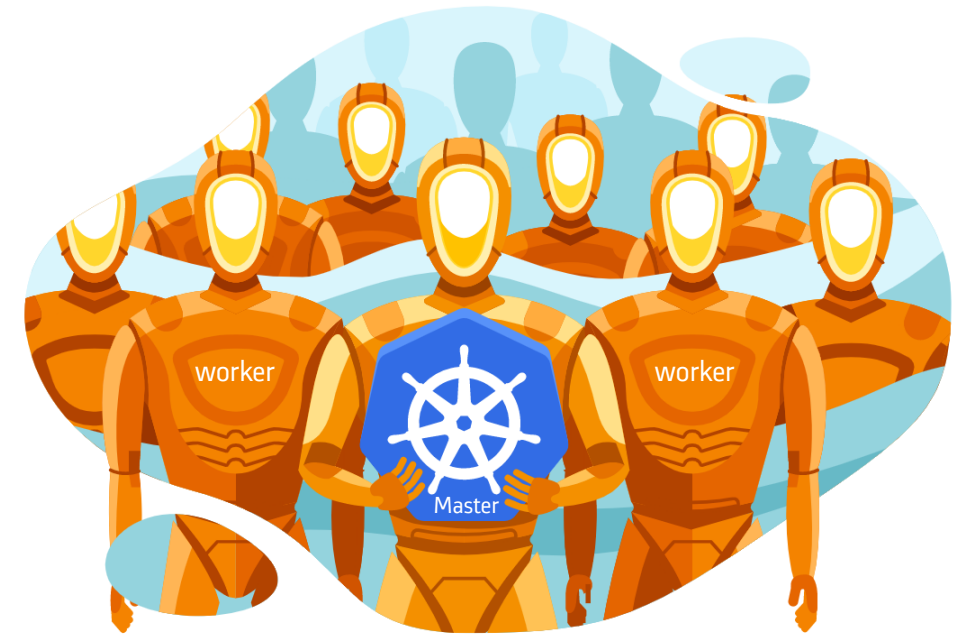
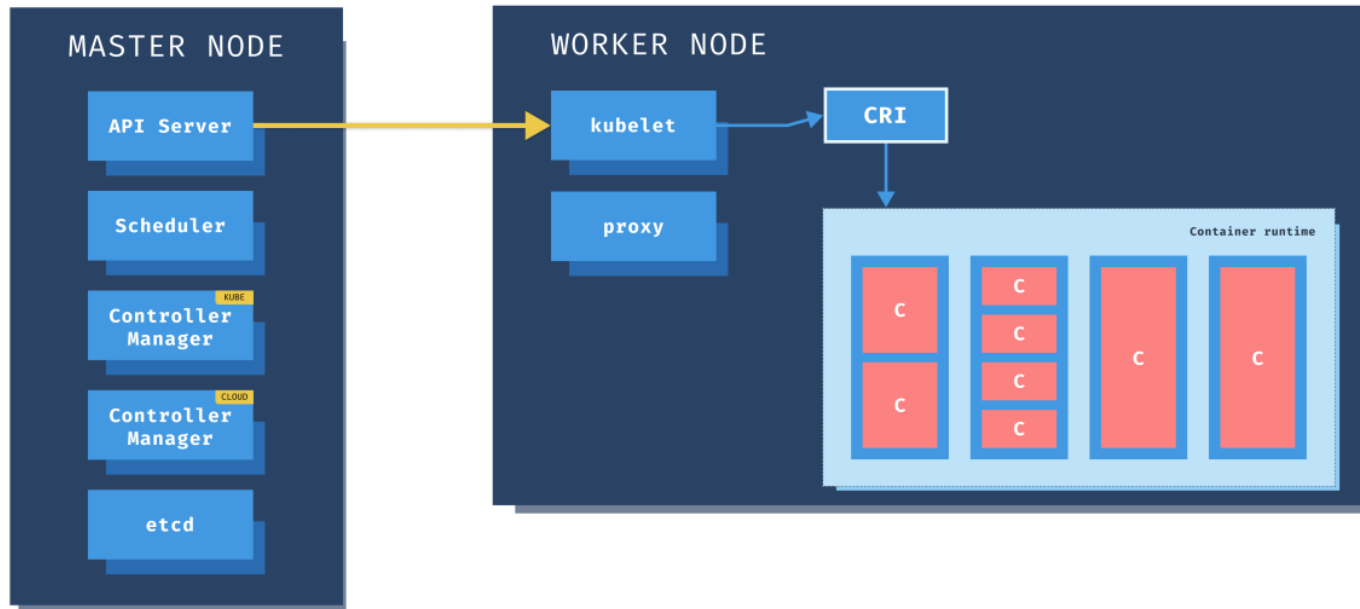
Kubernetes Cluster

A Kubernetes cluster is a set of physical or virtual machines and other infrastructure resources that are needed to run your containerized applications. Each machine in a Kubernetes cluster is called a **node**

There are two types of node in each Kubernetes cluster:

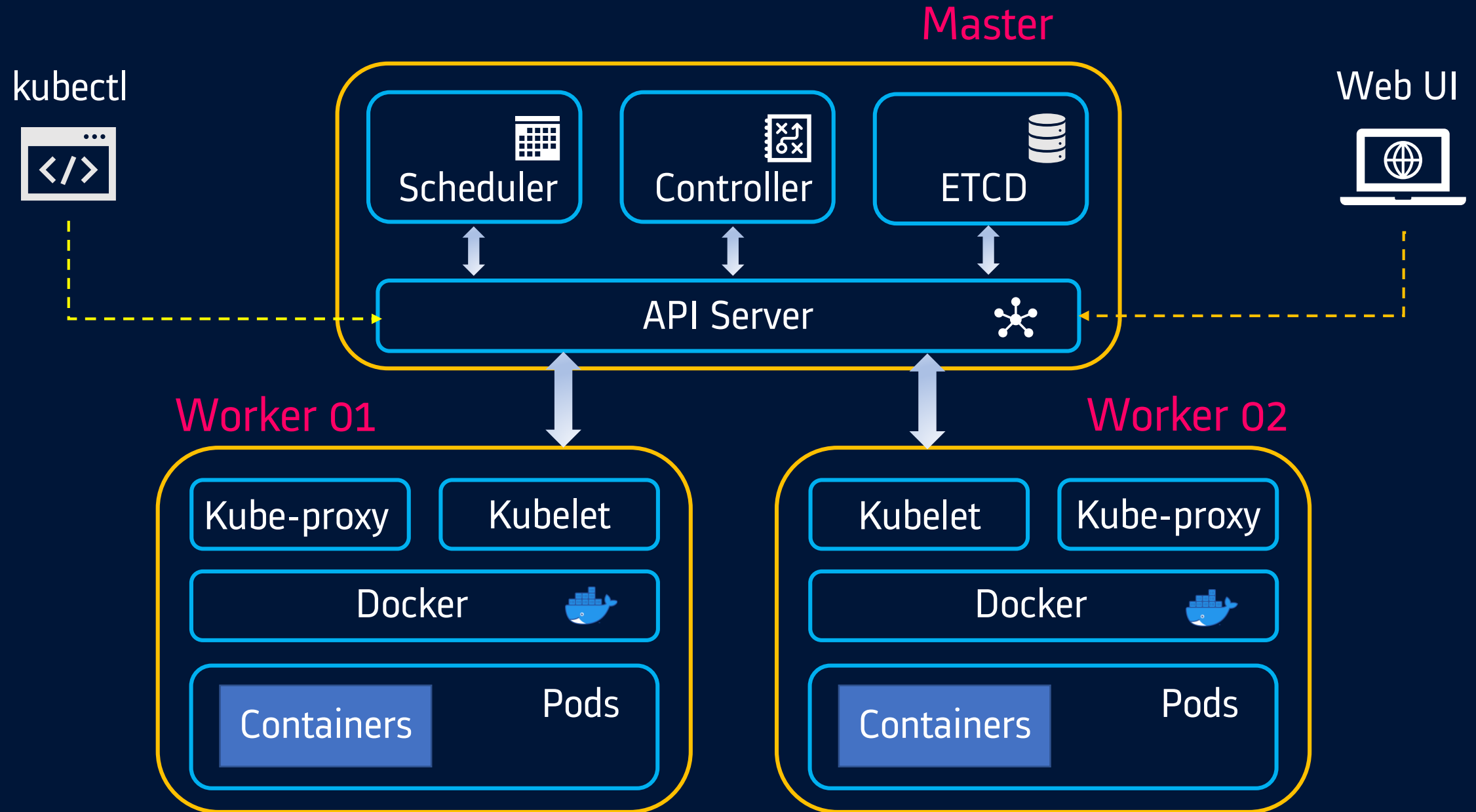
Master node(s): hosts the Kubernetes control plane components and manages the cluster

Worker node(s): runs your containerized applications





Kubernetes Architecture

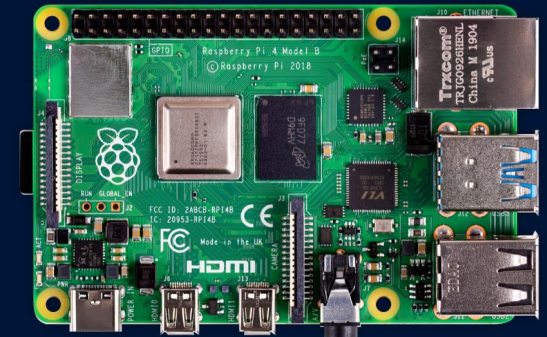




Kubernetes

What is K3s?

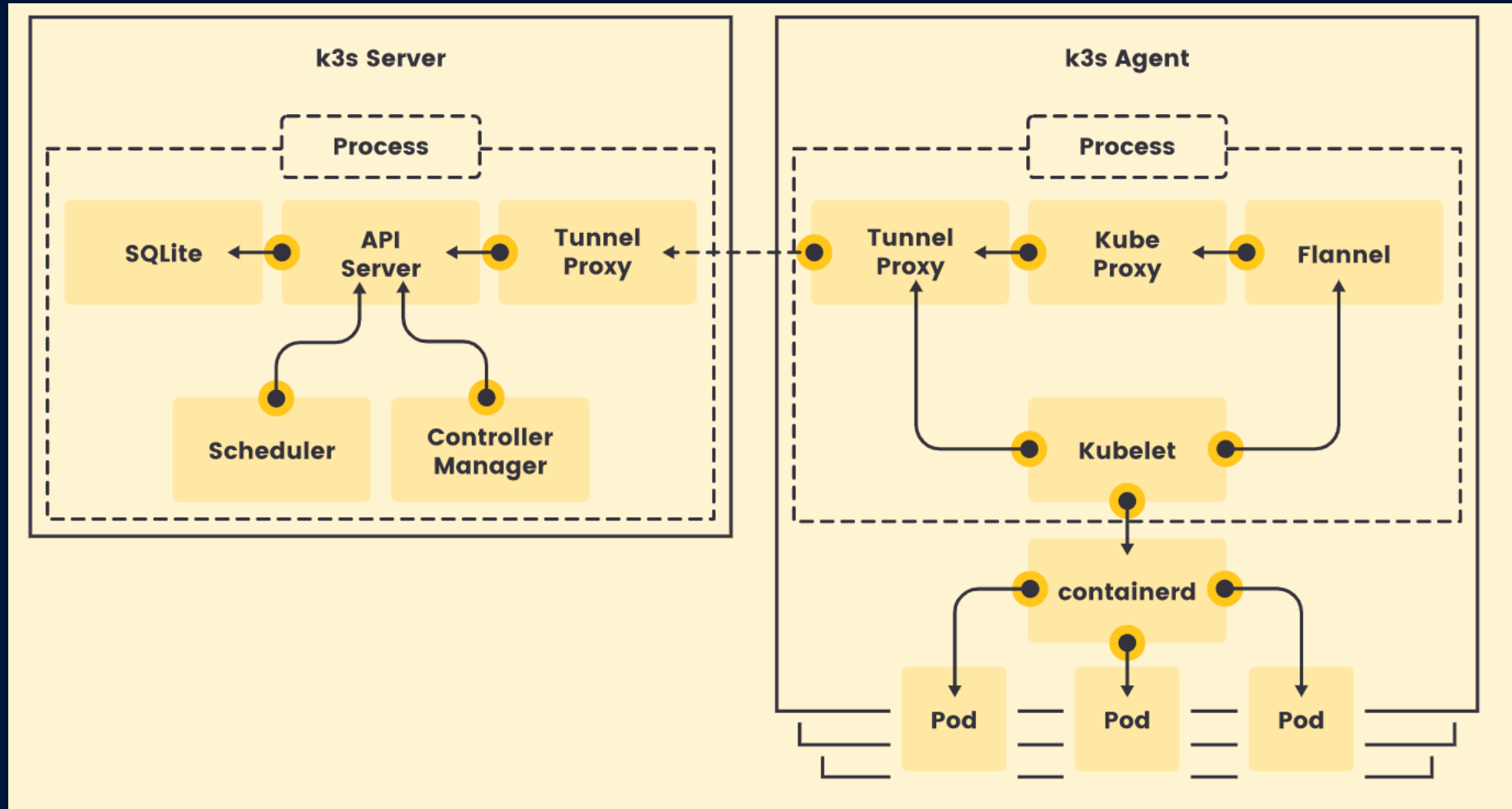
- **K3s** is a lightweight distribution of Kubernetes by Rancher that strips away a number of features while remaining fully compliant with up-stream Kubernetes
- It allows easier deployment when compared to **kubeadm** (BareMetal K8s setup tool) and all in a binary less than 40MB
- k3s is a fantastic solution for deploying Kubernetes on smaller devices, older hardware, and even IOT devices like **Raspberry Pi**
- **Key features:**
 - ✓ Packaged as a single binary
 - ✓ <40MB memory footprint
 - ✓ Supports ARM and x86 architectures
 - ✓ Lightweight storage backend based on **sqlite3** as the default storage mechanism to replace heavier **ETCD** server
 - ✓ **Docker** is replaced in favour of **containerd** runtime
 - ✓ Inbuilt **Traefik Ingress controller**
 - ✓ Inbuilt **metrics-server**





Kubernetes

K3s Architecture





| Kubernetes

K3s: What's in the name?

Kubernetes = K8s

K3s is designed to be half the size of a full blown Kubernetes implementation. Hence, the 5 letter K3S instead of 10 letter K8S





Kubernetes

K3s Cluster Setup using VirtualBox on Windows

- Use 3 VMs(1 master and 2 workers). All VMs should have bridge network adapter enabled
- Create a host only networking adapter(DHCP disabled) and connect all VMs to it. This is to have static IPs for all VMs in the cluster. Make sure static IPs are configured in each VM in the same subnet range of host only network
- Refer below link for cluster setup using VirtualBox on Windows
https://alstomgroup-my.sharepoint.com/:w:/p/vikram_babu-kunchala/EZRiFuT7qD9Pr6eXP-Lw9c0Bckfu5aBZhgcFCIMhPv24oA?e=oaYSh6

On Master Node

- Run `bash -c "curl -sfL https://get.k3s.io | sh -"`
- `TOKEN = cat /var/lib/rancher/k3s/server/node-token`
- `IP` = IP of master node where K8s API server is running

On Worker Nodes

- Run `bash -c "curl -sfL https://get.k3s.io | K3S_URL=\"https://$IP:6443\" K3S_TOKEN=\"$TOKEN\" sh -"`





Kubernetes

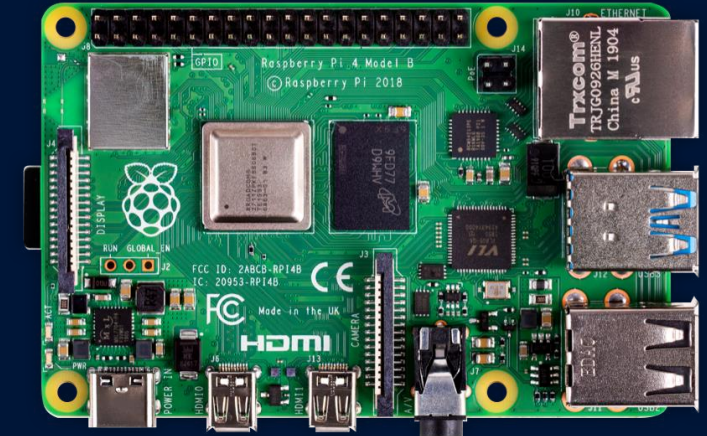
K3s single node cluster setup on Raspberry Pi

- Run following command to install K3s(requires internet access)

```
Terminal  
root@raspberrypi:/# curl -sfL https://get.k3s.io | sh -
```

- For offline installation

<https://rancher.com/docs/k3s/latest/en/installation/airgap/>





Kubernetes

Once the cluster is setup...

kubectl cluster-info

```
root@raspberrypi:/# kubectl cluster-info
Kubernetes master is running at https://127.0.0.1:6443
CoreDNS is running at https://127.0.0.1:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://127.0.0.1:6443/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy
```

kubectl version

```
root@raspberrypi:/# kubectl version --short
Client Version: v1.18.3+k3s1
Server Version: v1.18.3+k3s1
root@raspberrypi:/#
```

kubectl get nodes -o wide

```
root@raspberrypi:/# kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
raspberrypi	Ready	master	5m	v1.18.3+k3s1	192.168.0.100	<none>	Raspbian GNU/Linux 10 (buster)	4.19.118-v7+	containerd://1.3.3-k3s2

```
root@raspberrypi:/#
```

In case of multinode cluster you should see all the nodes in the cluster along with their IPs



Kubernetes

Metrics server

- K3s comes inbuilt with **Kubernetes Metrics server**, a cluster-wide aggregator of resource usage data
- It provides CPU & RAM usage statistics per node and per pod, via CLI

```
root@raspberrypi:/home/pi# kubectl get all --all-namespaces | grep -i metrics-server
kube-system      pod/metrics-server-7566d596c8-7p5hr      1/1      Running      1
kube-system      service/metrics-server                   ClusterIP 10.43.225.240 <none>
kube-system      deployment.apps/metrics-server           1/1      1            1
kube-system      replicaset.apps/metrics-server-7566d596c8 1/1      1            1
```

Usage:

kubectl top nodes

```
root@raspberrypi:/# kubectl top nodes
NAME          CPU(cores)   CPU%   MEMORY(bytes)  MEMORY%
raspberrypi   340m        8%     604Mi          65%
root@raspberrypi:/# █
```

kubectl top pods



Kubernetes

Running your first pod

```
kubectl run nginx --image=nginx --port=80
```

```
Terminal
root@raspberrypi:/# kubectl run nginx --image=nginx --port=80
pod/nginx created
```



kubectl get po -o wide

```
root@raspberrypi:/# kubectl get po -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP          NODE      NOMINATED NODE   READINESS GATES
nginx     1/1     Running   0           4m16s  10.42.0.9   raspberrypi  <none>            <none>
root@raspberrypi:/#
```

Pods get unique IP inside the cluster. This is private to the cluster and not accessible from outside

In K3s, taints on Master are removed by default. This allows us to schedule pods onto the master as well



Kubernetes

Running your first pod

Accessing the pod

```
root@raspberrypi:/# kubectl get po -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP        NODE
nginx   1/1     Running   0           4m16s  10.42.0.9  raspberrypi
root@raspberrypi:/#
```

```
root@raspberrypi:/# curl 10.42.0.9
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

- By default, pods are accessible within the cluster only.
- To expose them outside the cluster, use **services**



Kubernetes

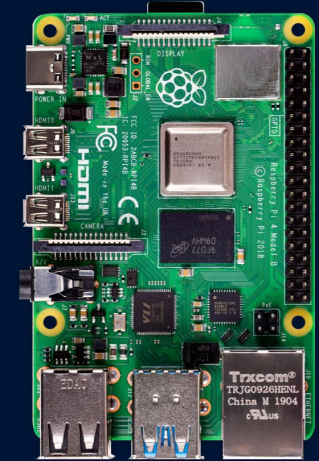
Creating a service

```
kubectl expose pod nginx --type=NodePort --name=nginx-service
```

```
Terminal

root@raspberrypi:/# kubectl expose pod nginx --type=NodePort --name=nginx-service
service/nginx-service exposed
```

192.168.0.100



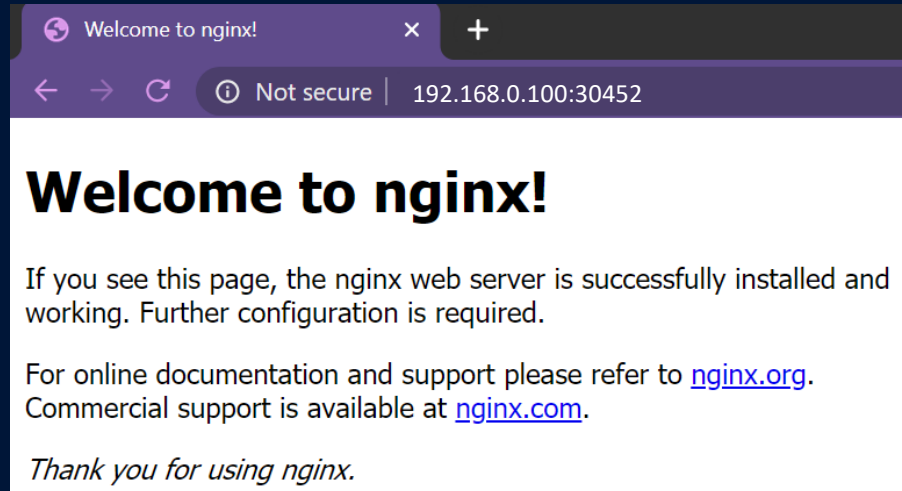
kubectl get service

```
root@raspberrypi:/# kubectl get service
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP   10.43.0.1     <none>         443/TCP          97m
nginx-service  NodePort    10.43.13.122  <none>         80:30452/TCP     4m14s
```

Now the application can be reached using

<node-ip>:<node-port>

192.168.0.100:30452





Kubernetes

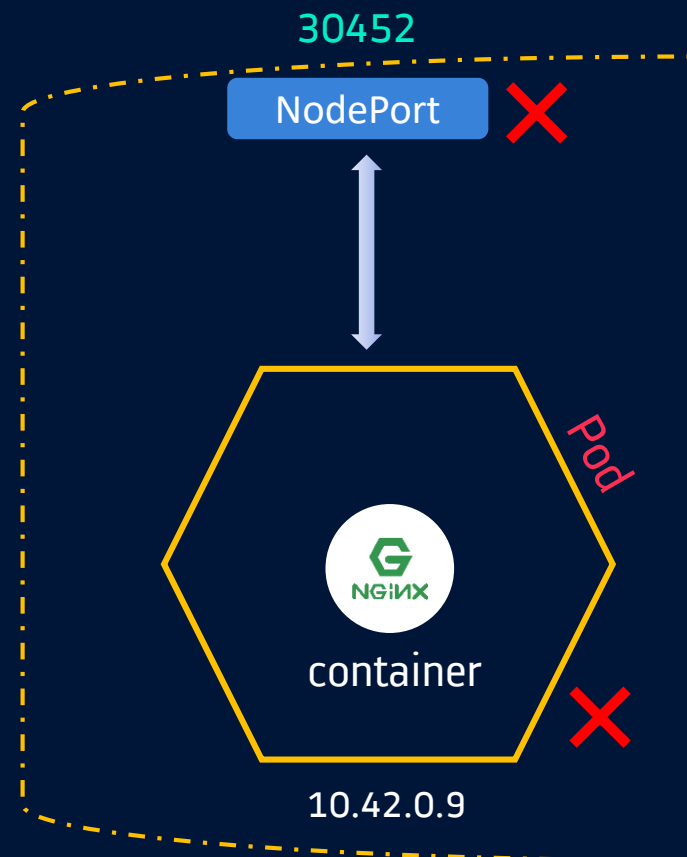
Deleting pods & services

kubectl delete pod nginx

kubectl delete service nginx-service

```
Terminal

root@raspberrypi:/home/pi# kubectl delete pod nginx
pod "nginx" deleted
root@raspberrypi:/home/pi# kubectl delete service nginx-service
service "nginx-service" deleted
root@raspberrypi:/home/pi#
```





Kubernetes

Inbuilt traefik ingress controller

- Traefik is an open source reverse proxy and load balancer for HTTP and TCP-based applications
- It is full-featured, production ready RP and LB and provides cluster network metrics

`kubectl get all --all-namespaces | grep -i traefik`

```
root@raspberrypi:/# kubectl get all --all-namespaces | grep -i traefik
kube-system      pod/helm-install-traefik-44nkc      0/1      Completed      0      109m
kube-system      pod/svclb-traefik-swwqj             2/2      Running        0      107m
kube-system      pod/traefik-758cd5fc85-sxmjq        1/1      Running        0      107m
kube-system      service/traefik-prometheus           ClusterIP  10.43.66.46    <none>     9100/TCP      107m
kube-system      service/traefik                      LoadBalancer 10.43.104.87  192.168.0.100 80:30605/TCP,443:32573/TCP 107m
kube-system      daemonset.apps/svclb-traefik         1         1             1         1         1         <none>      107m
kube-system      deployment.apps/traefik              1/1      1             1         1         107m
kube-system      replicaset.apps/traefik-758cd5fc85   1         1             1         1         107m
kube-system      job.batch/helm-install-traefik       1/1      100s         109m
root@raspberrypi:/#
```



- Traefik will generate a LoadBalancer IP automatically when the service is started



Kubernetes

Enable traefik dashboard service

- The dashboard is not enabled in the base k3s distribution
- Enable the dashboard by editing the `traefik.yaml` manifest at `/var/lib/rancher/k3s/server/manifests`
- Add the line `dashboard.enabled: "true"` to the `traefik.yaml` as shown in the screenshot
- Save the file and k3s will deploy the dashboard service

```
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: traefik
  namespace: kube-system
spec:
  chart: https://%{KUBERNETES_API}%/static/charts/traefik-1.81.0.tgz
  valuesContent: |-
    rbac:
      enabled: true
    ssl:
      enabled: true
    metrics:
      prometheus:
        enabled: true
    dashboard:
      enabled: true
    kubernetes:
      ingressEndpoint:
        useDefaultPublishedService: true
    image: "rancher/library-traefik"
    tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      - key: "node-role.kubernetes.io/master"
        operator: "Exists"
```

kubectl get service --all-namespaces

```
root@raspberrypi:/home/pi# kubectl get service --all-namespaces
NAMESPACE   NAME           TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)
default     kubernetes     ClusterIP     10.43.0.1    <none>        443/TCP
kube-system kube-dns       ClusterIP     10.43.0.10   <none>        53/UDP,53/TCP,9153/TCP
kube-system metrics-server ClusterIP     10.43.225.240 <none>        443/TCP
kube-system traefik-prometheus ClusterIP     10.43.66.46   <none>        9100/TCP
default     nginx-service  NodePort      10.43.13.122 <none>        80:30452/TCP
kube-system traefik      LoadBalancer 10.43.104.87  192.168.0.100 80:30605/TCP,443:32573/TCP
kube-system traefik-dashboard ClusterIP     10.43.204.133 <none>        80/TCP
root@raspberrypi:/home/pi#
```

within the cluster,
access the dashboard
using cluster-ip of
dashboard service




<http://10.43.204.133>



Kubernetes


Traefik dashboard

 **PROVIDERS** **HEALTH**

V1.7.19 / MAROILLES [DOCUMENTATION](#)

kubernetes

1 FRONTENDS

 traefik.example.com

MainDetails


Route Rule

Host:traefik.example.com


Entry Points

httphttps

Backend

 traefik.example.com

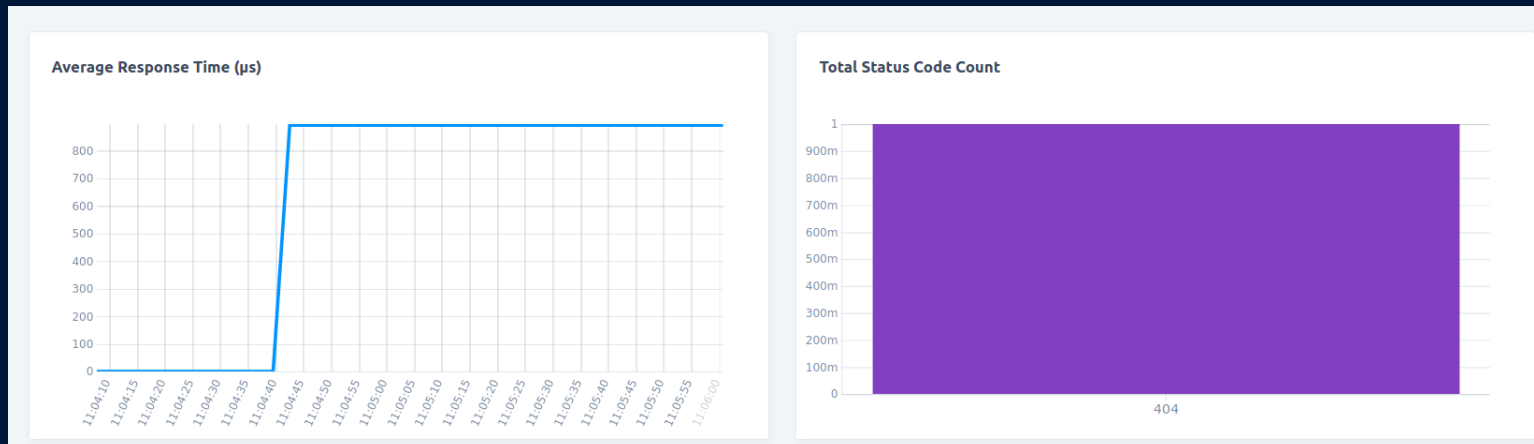
1 BACKENDS

 traefik.example.com

MainDetails

ServerWeight

<http://10.42.0.11:8080>1





Kubernetes

Kubernetes Dashboard service

kubectl apply -f

https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/dashboard/insecure-dashboard-clusterip.yaml

kubectl get services

```
root@proxyserver:/home/osboxes# kubectl get svc
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes          ClusterIP     10.43.0.1      <none>       443/TCP          5h22m
nodeport-nginx      NodePort      10.43.21.189   <none>       80:30378/TCP     23m
dashboard           ClusterIP     10.43.105.77   <none>       80/TCP           2m56s
root@proxyserver:/home/osboxes#
```

- within the cluster, access the dashboard using **cluster-ip** of dashboard service
- At the login page, click on 'skip' to access the dashboard



http://10.43.105.77

Kubernetes Dashboard

☒ Token
Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

☐ Kubeconfig
Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

Enter token *

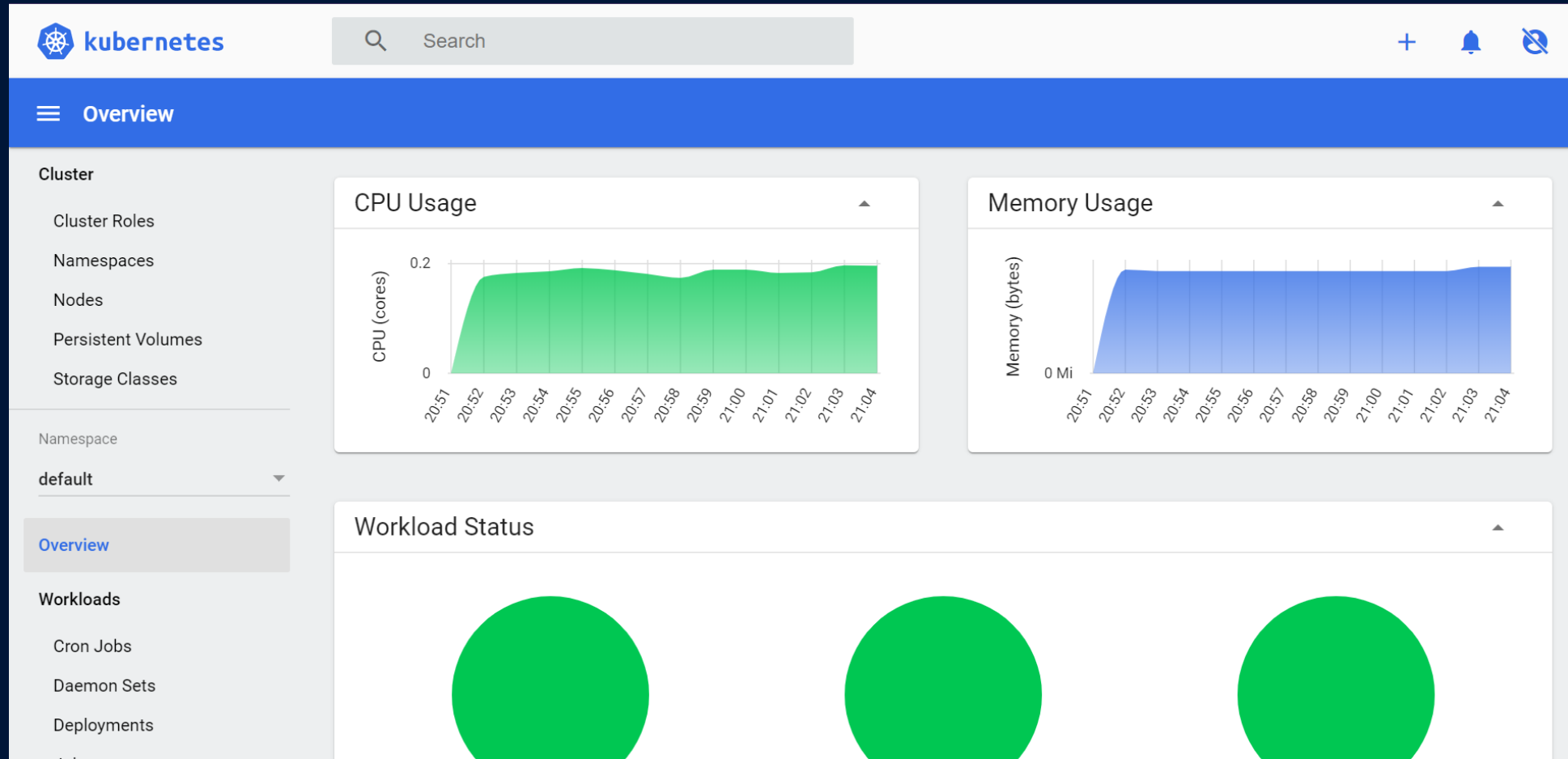


This is an insecure dashboard as secure login is bypassed in the manifest file. Only suitable for dev environments. In prod, login should be through a user token or kubeconfig file



Kubernetes

Kubernetes Dashboard service





Kubernetes

Ingress Controller

- Ingress resources cannot do anything on their own. We need to have an Ingress controller in order for the Ingress resources to work
- Ingress controller implements rules defined by ingress resources
- Ingress controllers doesn't come with all standard Kubernetes binary, they have to be deployed separately.
- Fortunately, K3s comes with Traefik Ingress controller inbuilt
- Kubernetes currently supports and maintains GCE and nginx ingress controllers
- Other popular controllers include HAProxy ingress, istio, Ambassador etc.,
- Ingress is the most useful if you want to expose multiple services under the same IP address
- Ingress controller can perform load balancing, Auth, SSL and URL/Path based routing configurations by being inside the cluster living as a Deployment or a DaemonSet

Kubernetes

Traefik Ingress Controller

 www.connected-city.com

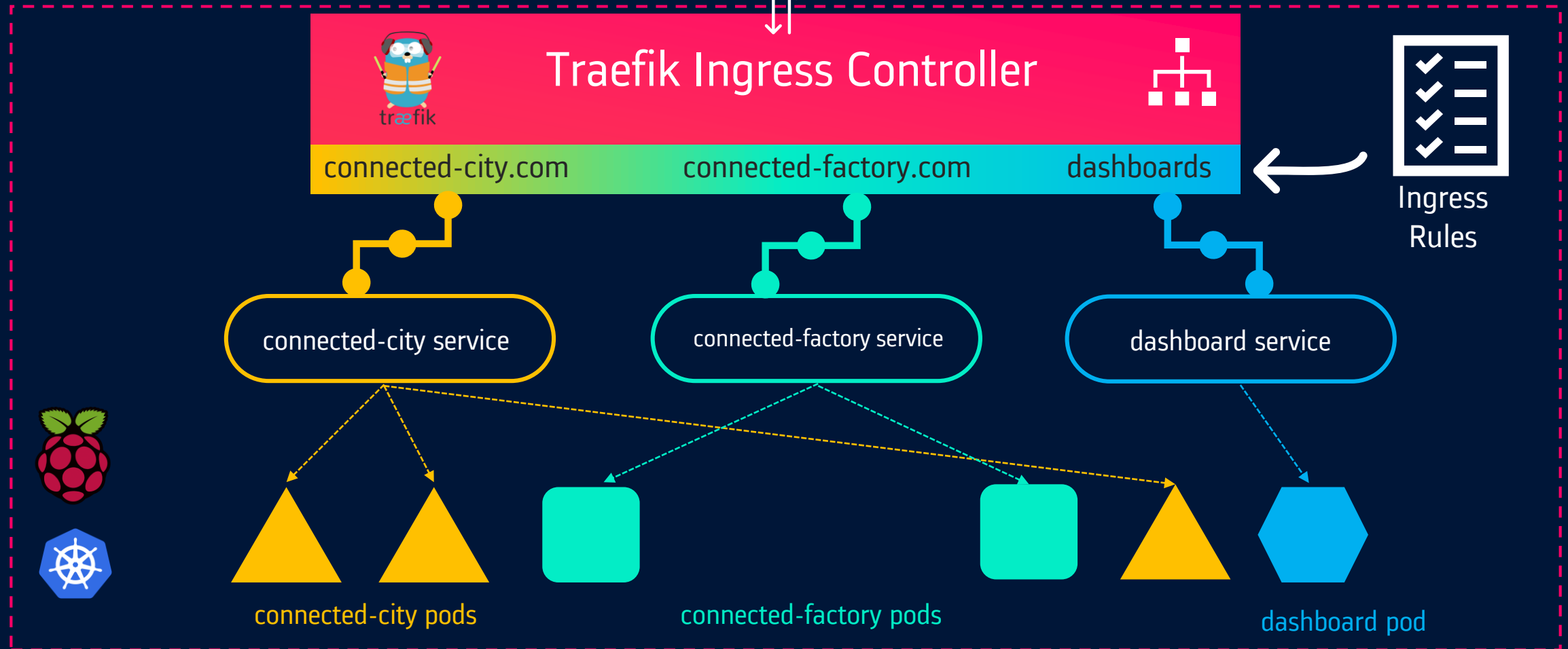
connected-city.com
connected-factory.com
traefik-dashboard.com
k8s-dashboard.com

DNS Resolution

192.168.0.100

Traefik LoadBalancer

Kubernetes Cluster/Single Node

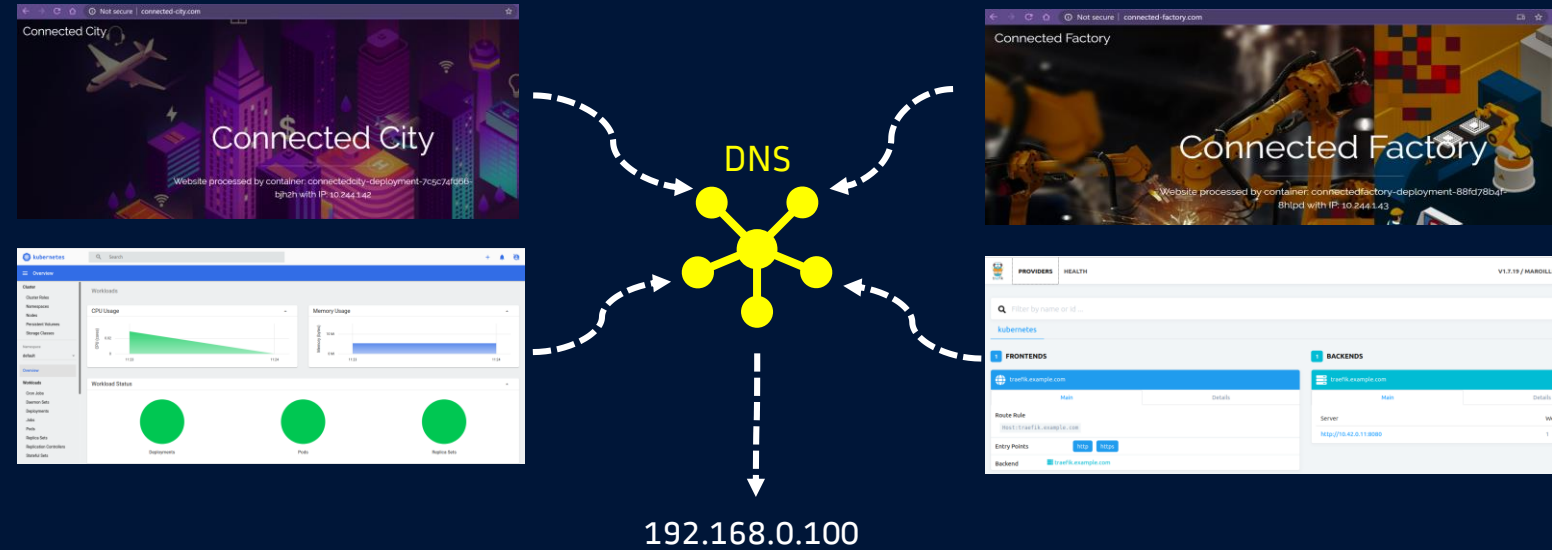




Kubernetes

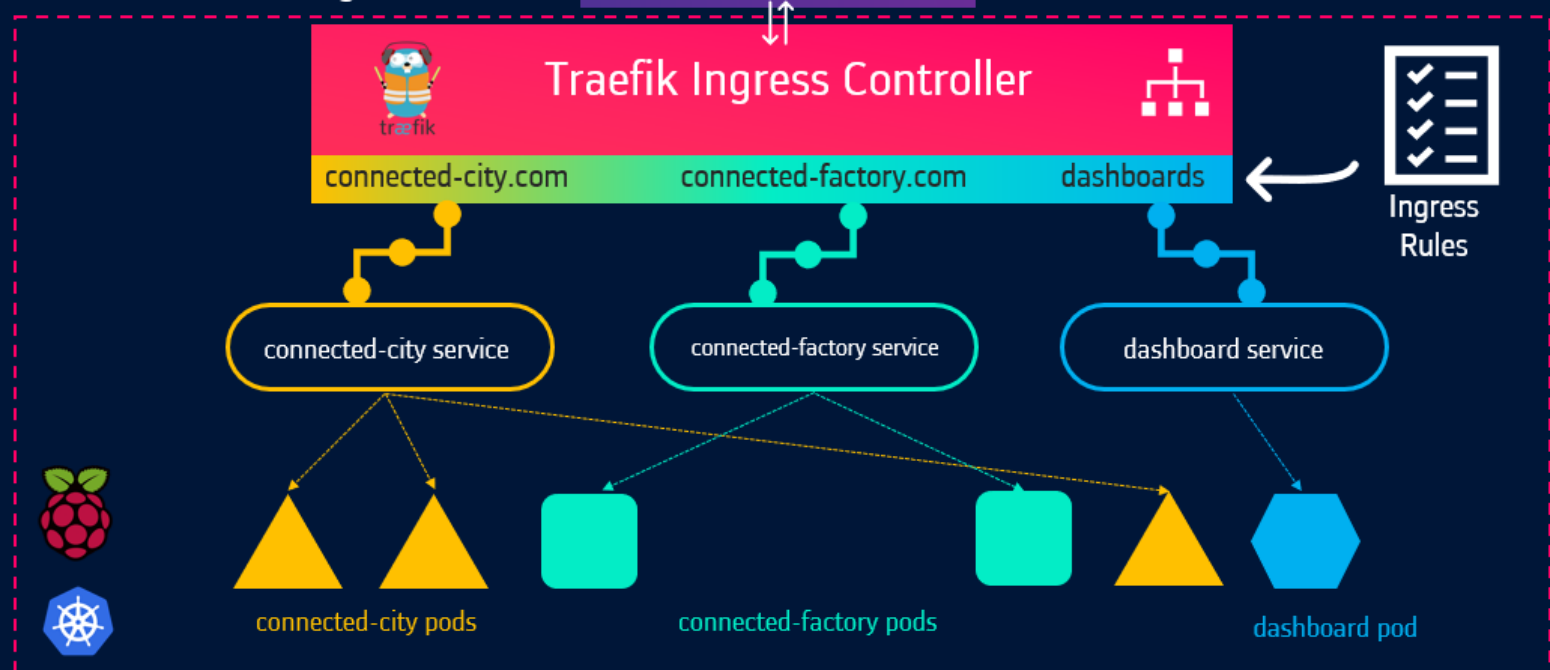
End-to-end demo

- 2 front-end applications
- K8s dashboard
- Traefik dashboard
- Ingress URL based routing
- All applications DNS resolution to same Traefik LoadBalancer IP



Kubernetes Cluster/Single Node

Traefik LoadBalancer





Kubernetes

End-to-end demo: Application Deployment

Deploy Kubernetes dashboard

`kubectl apply -f`

https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/dashboard/insecure-dashboard-clusterip.yaml

Deploy pods and services

`kubectl apply -f <object>.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: connectedcity-service
spec:
  ports:
    - port: 80
      targetPort: 5000
  selector:
    app: connectedcity
```

Application-1
Deployment + ClusterIP
service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: connectedcity-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: connectedcity
  template:
    metadata:
      labels:
        app: connectedcity
    spec:
      containers:
        - name: connectedcity
          image: kunchalavikram/connectedcity:v1
          ports:
            - containerPort: 5000
```

```
apiVersion: v1
kind: Service
metadata:
  name: connectedfactory-service
spec:
  ports:
    - port: 80
      targetPort: 5000
  selector:
    app: connectedfactory
```

Application-2
Deployment + ClusterIP
service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: connectedfactory-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: connectedfactory
  template:
    metadata:
      labels:
        app: connectedfactory
    spec:
      containers:
        - name: connectedfactory
          image: kunchalavikram/connectedfactory:v1
          ports:
            - containerPort: 5000
```



Kubernetes

End-to-end demo: Defining Ingress Rules

kubectl apply -f <object>.yaml

```

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-rules
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: connected-city.com
    http:
      paths:
      - backend:
          serviceName: connectedcity-service
          servicePort: 80
  - host: connected-factory.com
    http:
      paths:
      - backend:
          serviceName: connectedfactory-service
          servicePort: 80
  - host: k8s-dashboard.com
    http:
      paths:
      - backend:
          serviceName: dashboard
          servicePort: 80

```

when a request is received on the specific host URL, it is forwarded to the service mentioned.

To get services in all namespaces run

kubectl get svc -A

In this case, the traefik service is deployed in **kube-system** namespace. Hence the ingress rules should also be in the same namespace as the service.

```

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-rules-traefik-dashboard
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: traefik-dashboard.com
    http:
      paths:
      - backend:
          serviceName: traefik-dashboard
          servicePort: 80

```

* we can also use path based routing instead of URL based routing being shown in the demo



Ingress-controller executes these ingress-rules by comparing with the http requested URL in the http header



Kubernetes

End-to-end demo

Deploy complete Application with ingress rules using a single manifest file

`kubectl apply -f`

`https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress.yml`

```
root@raspberrypi:/home/pi# kubectl apply -f https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress.yml
ingress.networking.k8s.io/ingress-rules created
ingress.networking.k8s.io/ingress-rules-traefik-dashboard created
service/connectedcity-service created
deployment.apps/connectedcity-deployment created
service/connectedfactory-service created
deployment.apps/connectedfactory-deployment created
root@raspberrypi:/home/pi#
```

To get all objects inside the cluster like Pods, ReplicaSets, Deployments, Services...

`kubectl get all --all-namespaces`



Kubernetes

End-to-end demo

kubectl get pods --all-namespaces

```

root@raspberrypi:/home/pi# kubectl get po -A

```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	metrics-server-7566d596c8-7p5hr	1/1	Running	2	24h
kube-system	local-path-provisioner-6d59f47c7-rpvx5	1/1	Running	4	24h
kubernetes-dashboard	dashboard-metrics-scraper-dc6947fbf-cr2mp	1/1	Running	1	20h
kube-system	svclb-traefik-swwqj	2/2	Running	4	24h
kube-system	coredns-8655855d6-hz4rn	1/1	Running	2	24h
default	dashboard-8588744bfd-7qnkf	1/1	Running	1	20h
kubernetes-dashboard	kubernetes-dashboard-df6dbcbf8-lvxc8	1/1	Running	1	20h
kube-system	helm-install-traefik-z6csv	0/1	Completed	0	124m
kube-system	traefik-6cbfb44969-mlcgm	1/1	Running	0	123m
default	connectedfactory-deployment-88fd78b4f-wnxtk	0/1	CrashLoopBackOff	5	4m22s
default	connectedfactory-deployment-88fd78b4f-b9cpz	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-zb8g4	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-bn4fl	0/1	CrashLoopBackOff	5	4m22s
default	connectedfactory-deployment-88fd78b4f-5vfq4	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-vvczn	0/1	CrashLoopBackOff	5	4m22s

Pods fail to start
in RaspberryPi



kubectl logs pod/connectedfactory-deployment-88fd78b4f-b9cpz

```

root@raspberrypi:/home/pi# kubectl logs pod/connectedfactory-deployment-88fd78b4f-b9cpz
standard_init_linux.go:211: exec user process caused "exec format error"
root@raspberrypi:/home/pi#

```



Kubernetes

Docker Images for ARM

- If the applications were run on a regular PC or the x86_64 architecture, the pods would get created because the base image chosen to build the application images supports only those architectures
- Raspberry Pi hardware architecture is called **ARM** and differs from the architecture behind the regular PC, laptop or cloud instance. So regular docker images won't run in Pi
- To run docker containers in Pi, use only base images supported for ARM ----->

Ex: balenalib/raspberrypi3, arm32v6/python:3.5-alpine, arm32v6/alpine etc.,

```
root@raspberrypi:/home/pi# kubectl get po -A
```

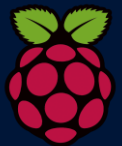
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	metrics-server-7566d596c8-7p5hr	1/1	Running	2	24h
kube-system	local-path-provisioner-6d59f47c7-rpvx5	1/1	Running	4	24h
kubernetes-dashboard	dashboard-metrics-scraper-dc6947fbf-cr2mp	1/1	Running	1	20h
kube-system	svclb-traefik-swwqj	2/2	Running	4	24h
kube-system	coredns-8655855d6-hz4rn	1/1	Running	2	24h
default	dashboard-8588744bfd-7qnkf	1/1	Running	1	20h
kubernetes-dashboard	kubernetes-dashboard-df6dbcbf8-lvxc8	1/1	Running	1	20h
kube-system	helm-install-traefik-z6cgv	0/1	Completed	0	124m
kube-system	traefik-6cbfb44969-mlcgm	1/1	Running	0	123m
default	connectedfactory-deployment-88fd78b4f-wnxtk	0/1	CrashLoopBackOff	5	4m22s
default	connectedfactory-deployment-88fd78b4f-b9cpz	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-zb8g4	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-bn4f1	0/1	CrashLoopBackOff	5	4m22s
default	connectedfactory-deployment-88fd78b4f-5vfq4	0/1	CrashLoopBackOff	5	4m22s
default	connectedcity-deployment-7c5c74fd66-vvczn	0/1	CrashLoopBackOff	5	4m22s

Dockerfile for x86_x64/AMD

```
FROM python:alpine3.7
COPY ./app
WORKDIR /app
RUN pip install flask
EXPOSE 5000
CMD python ./appv3.py
```

Dockerfile for ARM32

```
FROM arm32v6/python:3.5-alpine
COPY ./app
WORKDIR /app
RUN pip install flask
EXPOSE 5000
CMD python ./appv3.py
```





Kubernetes

End-to-end demo

Delete all previous deployment using kubectl delete command

`kubectl delete -f`

`https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress.yml`

```
root@raspberrypi:/home/pi# kubectl delete -f https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes\_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress.yml
ingress.networking.k8s.io "ingress-rules" deleted
ingress.networking.k8s.io "ingress-rules-traefik-dashboard" deleted
service "connectedcity-service" deleted
deployment.apps "connectedcity-deployment" deleted
service "connectedfactory-service" deleted
deployment.apps "connectedfactory-deployment" deleted
root@raspberrypi:/home/pi#
```




Kubernetes

End-to-end demo

Deploy Applications designed for ARM architecture

`kubectl apply -f`

https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress-arm32.yml

```
root@raspberrypi:/home/pi# kubectl apply -f https://raw.githubusercontent.com/kunchalavikram1427/Kubernetes_public/master/manifests/ingress/ingress-my-demo-with-k8s-traefik-ingress.yml
ingress.networking.k8s.io/ingress-rules created
ingress.networking.k8s.io/ingress-rules-traefik-dashboard created
service/connectedcity-service created
deployment.apps/connectedcity-deployment created
service/connectedfactory-service created
deployment.apps/connectedfactory-deployment created
root@raspberrypi:/home/pi#
```

`kubectl get pods -o wide`

All pods run in Pi as the images are for ARM architecture

```
root@raspberrypi:/home/pi# k get pods -o wide
NAME
dashboard-8588744bfd-7qnkf
connectedcity-deployment-76fddb88-lpc9w
connectedcity-deployment-76fddb88-zhn7l
connectedcity-deployment-76fddb88-ml9hc
connectedfactory-deployment-587f489dd7-kg46w
connectedfactory-deployment-587f489dd7-7gxc1
connectedfactory-deployment-587f489dd7-twths
root@raspberrypi:/home/pi#
```

READY	STATUS	RESTARTS	AGE	IP	NODE
1/1	Running	1	20h	10.42.0.25	raspberrypi
1/1	Running	0	98s	10.42.0.64	raspberrypi
1/1	Running	0	98s	10.42.0.60	raspberrypi
1/1	Running	0	98s	10.42.0.61	raspberrypi
1/1	Running	0	96s	10.42.0.63	raspberrypi
1/1	Running	0	96s	10.42.0.62	raspberrypi
1/1	Running	0	98s	10.42.0.65	raspberrypi



Kubernetes

End-to-end demo

Check metrics

kubectl top node

```
root@raspberrypi:/home/pi# kubectl top node
NAME          CPU(cores)   CPU%    MEMORY(bytes)  MEMORY%
raspberrypi   923m         23%     695Mi          75%
root@raspberrypi:/home/pi#
```

Checking Ingress rules in all namespaces

kubectl get ingress -A

```
root@raspberrypi:/home/pi# kubectl get ingress -A
NAMESPACE   NAME                               CLASS      HOSTS                                                                 ADDRESS          PORTS   AGE
kube-system  traefik-dashboard                 <none>     traefik.example.com          192.168.0.100   80      136m
default      ingress-rules                     <none>     connected-city.com,connected-factory.com,k8s-dashboard.com  192.168.0.100   80      2m48s
kube-system  ingress-rules-traefik-dashboard  <none>     traefik-dashboard.com        192.168.0.100   80      2m48s
root@raspberrypi:/home/pi#
```



Kubernetes

End-to-end demo

Service endpoints

kubectl describe ingress ingress-rules

```
root@raspberrypi:/home/pi# kubectl describe ingress ingress-rules
Name:                ingress-rules
Namespace:           default
Address:             192.168.0.100
Default backend:     default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
Rules:
  Host                Path    Backends
  ----                -
  connected-city.com  /       connectedcity-service:80 (10.42.0.60:5000,10.42.0.61:5000,10.42.0.64:5000)
  connected-factory.com /       connectedfactory-service:80 (10.42.0.62:5000,10.42.0.63:5000,10.42.0.65:5000)
  k8s-dashboard.com   /       dashboard:80 (10.42.0.25:80)
Annotations:         kubernetes.io/ingress.class: traefik
Events:              <none>
root@raspberrypi:/home/pi#
```

In this case, when request comes from the URL, it is forwarded to the **ClusterIP** service inside the cluster. Since each application has 3 **replicas**, the service Load Balances the requests among all the available backend pods, which are shown above with their IPs and Container ports. For this demo, flask containers are used and hence the default port 5000



Kubernetes

End-to-end demo

Service endpoints

`kubectl describe ingress ingress-rules-traefik-dashboard -n kube-system`

```
root@raspberrypi:/home/pi# kubectl describe ingress ingress-rules-traefik-dashboard -n kube-system
Name:          ingress-rules-traefik-dashboard
Namespace:     kube-system
Address:       192.168.0.100
Default backend: default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
Rules:
  Host          Path  Backends
  ----          -
  traefik-dashboard.com
  traefik-dashboard.com      traefik-dashboard:80 (10.42.0.52:8080)
Annotations:    kubernetes.io/ingress.class: traefik
Events:         <none>
root@raspberrypi:/home/pi#
```

Here the traefik-dashboard service is deployed in `kube-system` namespace and hence the ingress rules are also deployed in the same kube-system namespace.

We have only 1 instance of traefik dashboard running in the cluster. So there is only one endpoint (10.42.0.52:8080)



Kubernetes

End-to-end demo

Update dummy DNS entries

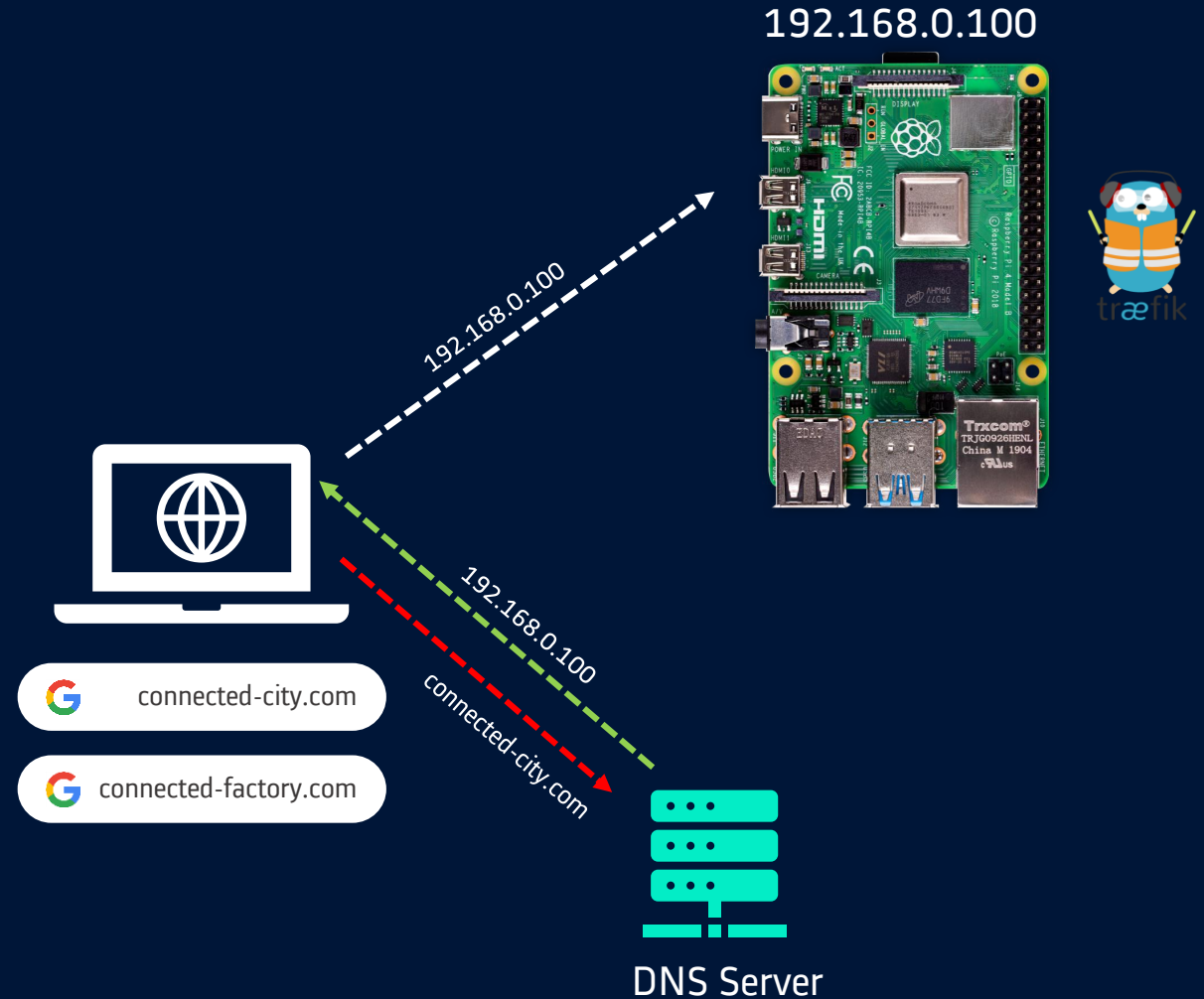
Update DNS names to point to LoadBalancer IP of Traefik Ingress Controller

windows

```
C:\Windows\System32\drivers\etc\hosts
192.168.0.100 connected-city.com
192.168.0.100 connected-factory.com
192.168.0.100 k8s-dashboard.com
192.168.0.100 traefik-dashboard.com
ipconfig /flushdns
```

linux

```
/etc/hosts
192.168.0.100 connected-city.com
192.168.0.100 connected-factory.com
192.168.0.100 k8s-dashboard.com
192.168.0.100 traefik-dashboard.com
```



* For RaspberryPi or the cluster



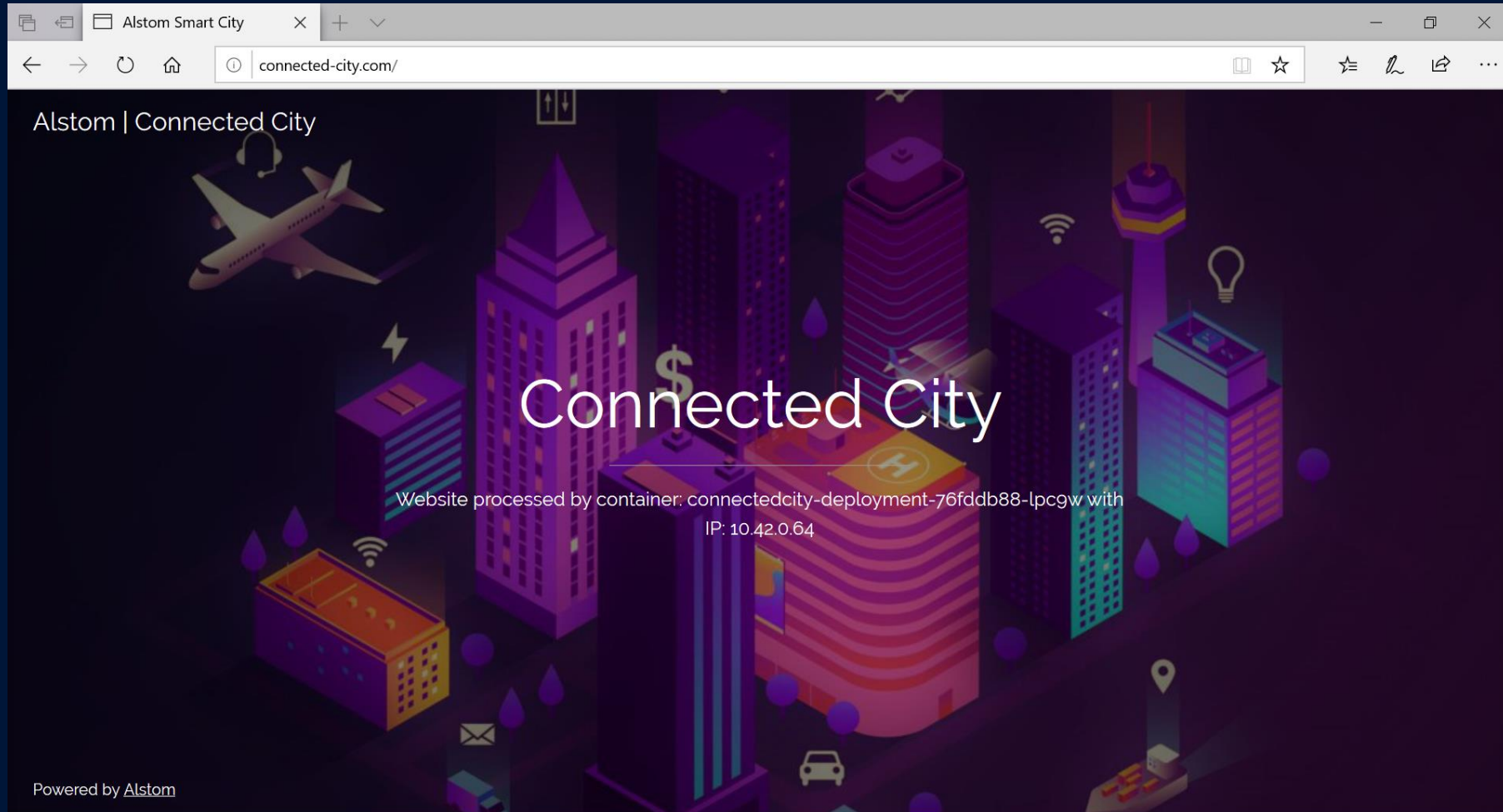
Kubernetes

End-to-end demo

Access Application through URLs



connected-city.com




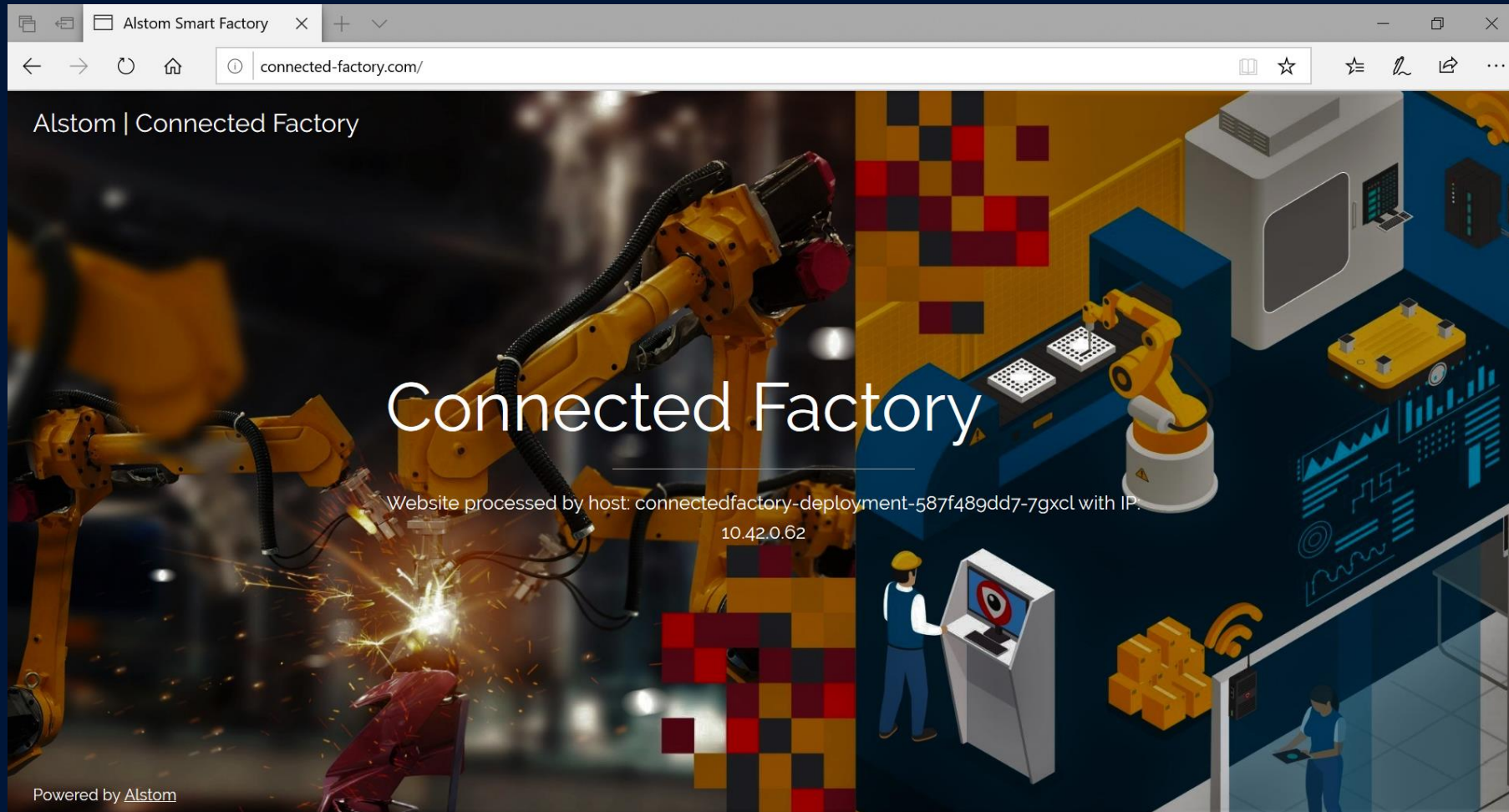


Kubernetes

End-to-end demo

Access Application through URLs

 connected-factory.com





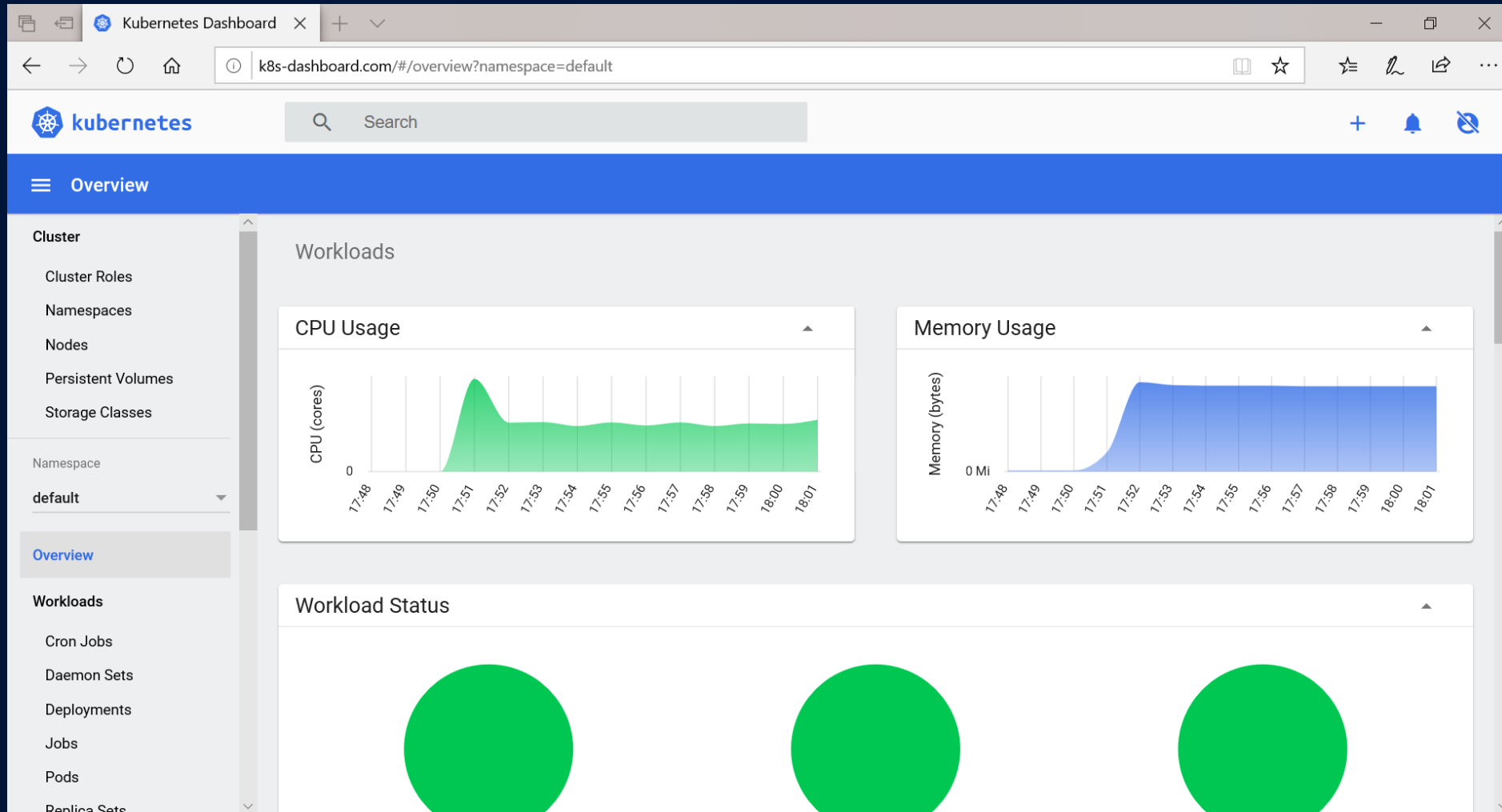
Kubernetes

End-to-end demo

Access Application through URLs



k8s-dashboard.com



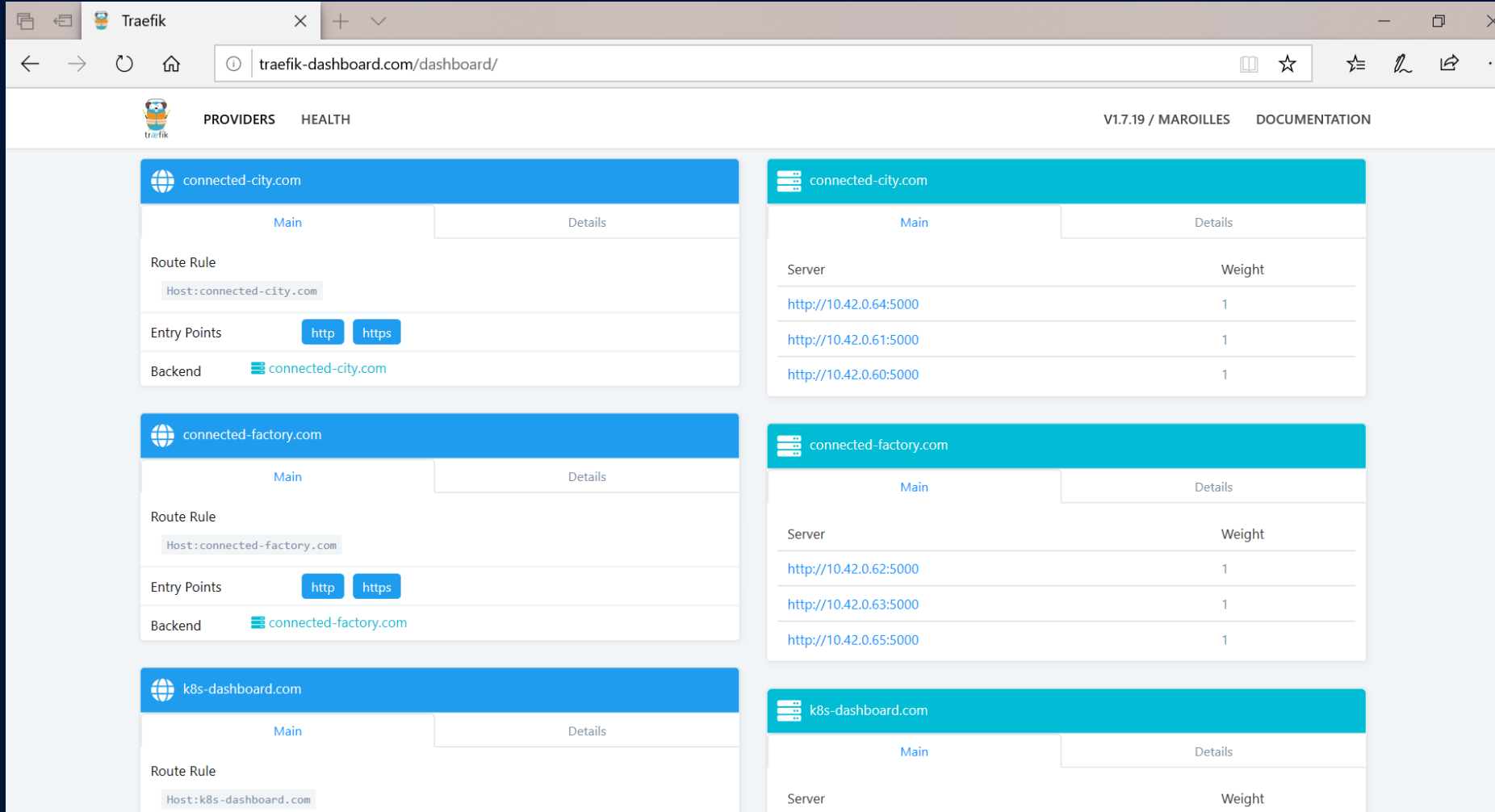


Kubernetes

End-to-end demo

Access Application through URLs

 traefik-dashboard.com



The screenshot shows the Traefik dashboard interface. The top navigation bar includes the Traefik logo, 'PROVIDERS', 'HEALTH', and version information 'V1.7.19 / MAROILLES' with a 'DOCUMENTATION' link. The main content area displays three service configurations, each with a 'Main' and 'Details' tab.

connected-city.com

- Route Rule: Host: connected-city.com
- Entry Points: http, https
- Backend: connected-city.com

connected-factory.com

- Route Rule: Host: connected-factory.com
- Entry Points: http, https
- Backend: connected-factory.com

k8s-dashboard.com

- Route Rule: Host: k8s-dashboard.com

connected-city.com Details

Server	Weight
http://10.42.0.64:5000	1
http://10.42.0.61:5000	1
http://10.42.0.60:5000	1

connected-factory.com Details

Server	Weight
http://10.42.0.62:5000	1
http://10.42.0.63:5000	1
http://10.42.0.65:5000	1

k8s-dashboard.com Details

Server	Weight
--------	--------



Kubernetes

End-to-end demo

Access Application through URLs

 traefik-dashboard.com

