

Running k8s on Raspberry Pi's

Running k8s on Raspberry Pi's



Summary

- **What is kubernetes and why is kubernetes**
- **Raspberry Pi's are totally cool**
- **Deploying k3s kubernetes on Raspberry Pis**
- **Setting up the WebUI as your first deployment**
 - Having a look at the different things in the WebUI
- **Deploy more things in your Cluster**
 - kubectl and helm
- **Some notes about deploying your own stuff in your Cluster**
 - A note on ARM and container architectures
 - A note on Storage

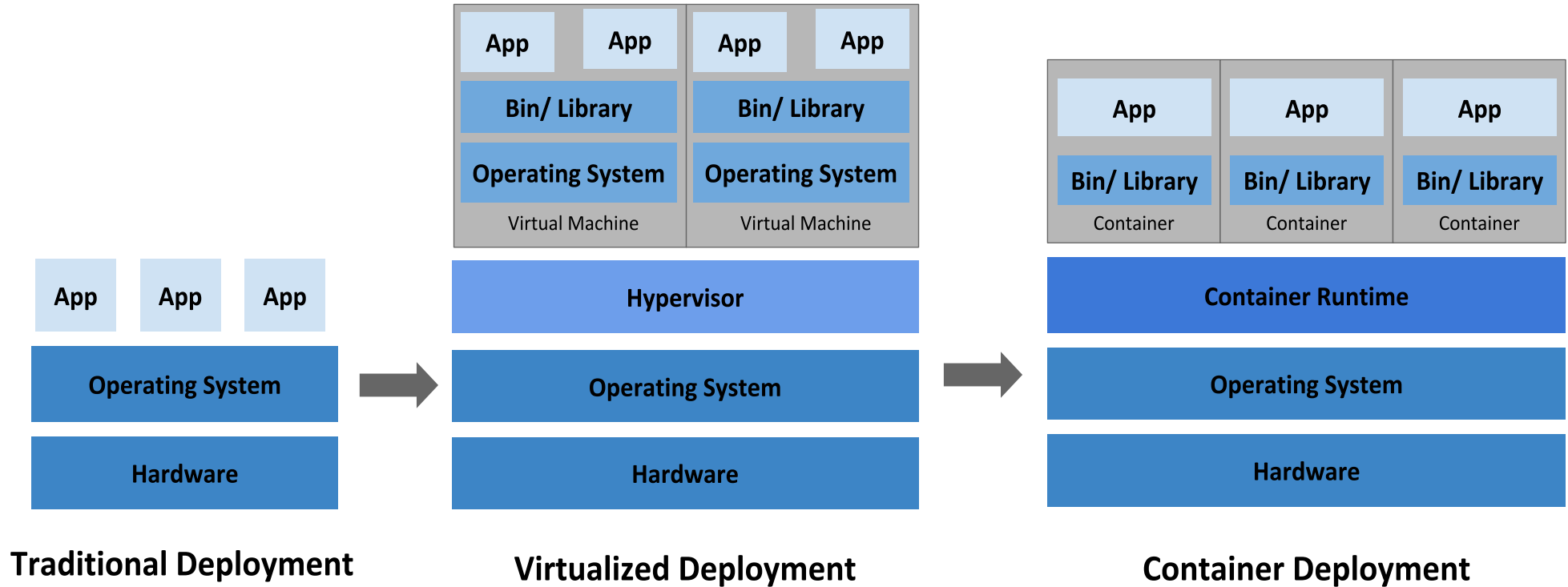


What we will not cover

- **How to work with Kubernetes**
- **Kubernetes-in-depth**



Little Bit of History of Virtualization



Containers at enterprise scale

- **When you have a lot of these, you need a guide**
- **Something to orchestrate it all**
- **Something to make your collection of containers bigger than the sum of it's parts**



What the Cloud is about

- **The Cloud is NOT about cost control**
 - Quite the opposite in practice
- **The Cloud is about flexibility and scalability**
- **You buy the load, and rent the spike**
 - Pets vs Cattle



Intro to Kubernetes: What is it?



- aka **k8s** (k-ubernete(8)-s)
 - Kubernetes: Greek for pilot/captain/helmsman: he who guides
- **Comes out of Google**
 - Now run by the Cloud Native Computing Foundation (CNCF)
- <https://www.kubernetes.io>



What does it do?

- **Kubernetes is about containers running ‘services’**
 - It’s gotta be a container! (which doesn’t mean it’s gotta be docker)
 - Mostly micro-services and web applications
 - Anything that can be turned into a container, including batch jobs
- **It’s about running them efficiently**
 - Service discovery
 - Load balancing
 - Storage
 - Continuous optimization of available resources
- **And to make sure they keep running**
 - Even when one of your servers/cattle *disappear*



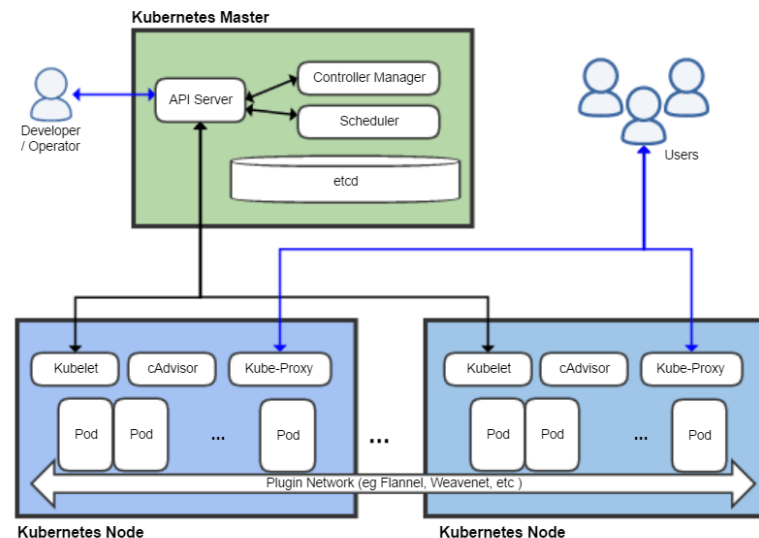
How to think of it

- **Sorta similar to docker-swarm, but also very much not**
- **It's not limited to container orchestration, deployment, management, scaling, storage, etc...**
- **It's about resource management and ancillary artifact management as well**
- **You declare the 'what' and it takes care of the 'how'**
 - What: Containerized workloads consisting of n parts declared in a YAML file
 - How: Which piece of your 'what' goes where based on resource availability?
- **In the last years, has become the *de facto* standard for micro-service and web application deployment**
 - e.g. RedHat OpenShift is running on top of Kubernetes



Digging In: What does a k8s cluster look like

- **A 'master' (main) node**
 - Little note on terminology here, in our current day and age
- **One or more worker nodes**
- **Main node combines the workers into a logical single unit, and runs**
 - API server to manage the cluster; note: not a dashboard (but we'll get to that)
 - Kubectl talks to this API server
 - A scheduler, to allocate 'resources' to 'workloads'
 - A controller manager: node, replication, endpoint, service accounts and tokens
 - etcd for configuration storage
 - 'cloud controller manager'
 - This is only for if you run this in the cloud to enable automatic provisioning
 - Or if you do some fancy stuff
- **Workers nodes are 'resources' to the cluster, each runs**
 - Kubelet: the cluster's residency inside each node
 - cAdvisor: Container Advisor monitoring resource usage
 - Kube-Proxy, to access the services running in *pods* on the node
 - This is what users interact with and routes their requests to their actual node on which the workload is running
 - Container runtime (typically docker), running your actual workloads in 'pods', which are groupings of containers that must run together on the same node
- **All glued together over IP**
- **In a way, worker nodes are ephemeral, they are just a resource to the cluster**
 - They can come
 - They can go
 - They can stay



Kubernetes and the Cloud

- **As such, Kubernetes is just a fabric**
 - Doesn't dictate much about what runs inside
- **While not PaaS as such, definitely offered as PaaS**
 - GCP GKE: Google Kubernetes Engine
 - AWS EKS: Elastic Kubernetes Service
 - Azure AKS: Azure Kubernetes Service
 - Let provider run your 'fabric', and you care about the apps
 - Very feudal-lord-y, mind you, but that's just me
 - This fits into the buy-the-load-rent-the-spike story
 - Add nodes when needed, release them when no longer needed



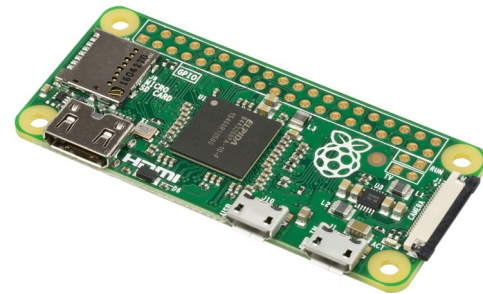
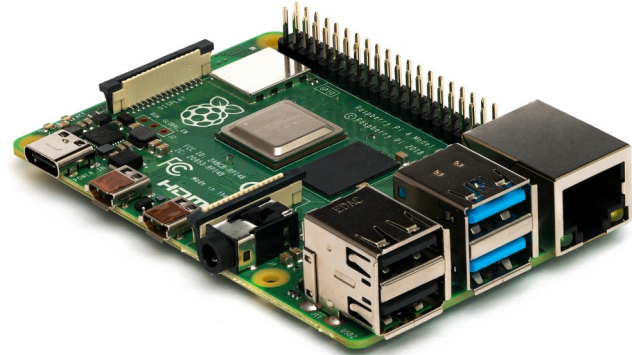
ELMO

- **Enough with the marketing and background**
- **Let's Move On**



Little bit of a plug again...

- **If you don't have Raspberry Pi's, go and get some!**
 - Seriously, super cool things
 - I love these things to death
- **Especially Raspberry Pi 4s**
 - 2/4/8 GB RAM, 1.5GHz 64-bit Quad-Core ARM CPU, 2(!) HDMI Out, 2x USB2, 2x USB3
 - ~ 100 USD for the Pi4 4GB version
- **There's even a Raspberry Pi zero**
 - 5 USD, 512 MB RAM, 1 HDMI, 1GHz 32-bit ARM CPU
 - ~ 5 USD
- **Go get some!**
 - <https://www.raspberrypi.org/>
 - MicroCenter, ...
- **Little bit of a caveat**
 - **ARM** architecture, not x86_64 (remember that)



My thinking...

- **Hardware hanging around**

- 9 Raspberry Pi's add up to 32 CPUs + 30GB RAM
 - I lie, one is dedicated to XBian (thanks to Tim Spangler :))
- Quite some compute power there
- All networked doing ... little (up and until recently)

- **I know, let's join these together using Kubernetes!**

- **And voila, you're own little scalable Cloud, running in a cupboard (I mean server room... definitely server room!)**

- Mentality: Buy Pi's to cover the load, buy more Pi's to cover the spike
 - Then find out how to turn the spike into the load
 - Thus justifying to your partner your desire to buy even more Pi's covering a future spike
 - This can't fail...
 - Can it?



Kubernetes on Raspberry Pi

- **Problem: Kubernetes runs on x86_64**
- **But there's something called k3s**
 - By Rancher: <https://www.k3s.io>
 - “Kubernetes for IoT and Edge computing”
 - Certified (by CNCF) k8s distribution, i.e. it's ‘real k8s’
 - Optimized for ARM
 - Ooohhhhh... now you have my attention!



Materials Needed

- **1 Raspberry Pi as Main Node**
- **1+ Raspberry Pi's as Worker Nodes**
 - Note: your Main node also runs workloads
- **All nodes running plain Raspbian (Debian)**
 - On your network
 - With ability to SSH in
- **Ansible, because why not**
 - Or SaltStack (Nick Vissari)
 - If you want, I can share my Ansible Playbooks



Step 1: Install and enable Docker on all nodes

- **Remove old docker packages if they are there**
 - apt install containerd docker docker-engine docker.io runc
- **Install docker**
 - apt -no-install-recommends install containerd.io docker-ce docker-ce-cli
 - no-recommends because there's some issues with advanced multi-layered unification filesystem (AUFS)
- **Explicitly get rid of AUFS**
 - apt remove aufs-dkms
- **Start and enable docker**
 - systemctl start docker
 - systemctl enable docker
- **Firewall stuff: Make sure FORWARD is set to ACCEPT**
 - iptables -P FORWARD ACCEPT
 - This is needed because of kube-proxy
- **If needed, make your user part of the docker group**
 - usermod -aG \${USER} docker



Step 2.a: Install k3s on the main node

- **Create a group 'kubernetes' and join it**
 - You'll need this later for 'helm' stuff...
 - `groupadd kubernetes`
 - `usermod -aG ${USER} kubernetes`
- **Pipe the script into a shell (don't get me started)**
 - `curl -sL https://get.k3s.io | sh -`
- **Grab the server token for the main node**
 - This is a sensitive value!
 - `cat /var/lib/rancher/k3s/server/node-token`
 - Something like `Khexhex....:server:hexhexhex`
 - Base64-decode it!
- **Cool, your main node is up!**



Step 2.b: Sanity check

- **SCP**

- scp "main_node:/etc/rancher/k3s/k3s.yaml" "~/.kube/config"
- YAML file
- Change clusters.cluster[name=default].server to the name or IP of your main node
- Save this in ~/.kube/config

- **Download kubectl**

- <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- pushd ~/bin
- curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s <https://storage.googleapis.com/kubernetes-release/release/stable.txt>`/bin/linux/amd64/kubectl
- chmod +x ./kubectl
- Popd

- **Talk to your cluster, whisper in its ear**

- kubectl get node

NAME	STATUS	ROLES	AGE	VERSION
rpi40	Ready	master	179d	v1.18.4+k3s1



Step 3.a: Add worker nodes

- **Pipe the script into a shell**

- Make sure the following environment variables are set:
 - K3S_URL=https://main_node_ip:6443
 - K3S_TOKEN=server token (base64 decoded, right?)
- `$ K3S_URL=".." K3S_TOKEN=".." curl -sfL \`
`https://get.k3s.io | sh -`

- **Cool, your worker(s) are up**

- And part of your cluster



Step 3.b: Sanity check

- **Make sure your cluster sees your nodes**

- `kubectl get node`

```
$ kubectl get node
```

NAME	STATUS	ROLES	AGE	VERSION
rpi40	Ready	master	179d	v1.18.4+k3s1
rpi41	Ready	<none>	179d	v1.18.4+k3s1

...



Interludium: Adding ‘stuff’ to your cluster

- **So you’ve got a cluster... cool, but it’s empty**
- **Things you’d add**
 - Namespaces: scopes for workloads, customers, etc...
 - Almost everything is scoped within a namespace, handful of things aren’t
 - Namespaces can have resource constraints defined
 - Workloads: applications consisting of containers (e.g. WebUI, GitLab runner)
 - Deployments, Pods, Cron Jobs, ...
 - Services, Ingress
 - Storage configurations
 - Security ‘stuff’: roles, users, tokens, secrets
- **2 non-exclusive Ways**
 - Sometimes, you define everything yourself in a YAML file and upload through kubectl
 - You already downloaded that
 - `$ kubectl apply -f file.yaml`
 - Sometimes you use ‘helm’: <https://helm.sh/docs/intro/install/>
 - Helm (cf. “Helmsman”) is to Kubernetes as apt is to Debian, and works very similarly too
 - cf. Repositories
 - Bunch of Helm solutions already defined as templates in which you “just fill in some variable values”
 - e.g. GitLab Runner
 - You can run Helm on your main node, or on your own machine and point it at your cluster to do things in
 - This depends on what it is you’re doing



Step 4.a: Add web UI

- **WebUI is the dashboard for Kubernetes**

- Deployed through a YAML file and applied through kubectl
 - kubectl apply -f kubernetes-dashboard.yaml kubernetes-users.yaml
 - kubernetes-users.yaml creates a dashboard-admin (which you'll need in a bit) in kubernetes-dashboard namespace
 - Does RBAC 'stuff' and deploys the dashboard

- **This is where things get a little complicated**

- Because now we're applying files
- And it's a pretty complicated file, so
 - Either I wave my hands and go "Trust me"
 - Or we dive in together, and then you're in for something...
 - Maybe at the end?



Step 4.b: Inspect your Dashboard

- We need to get some credentials to be able to access the dashboard, we'll use a bearer token
- Grab your token which was installed when you deployed the Dashboard

```
$ kubectl get secret -n kubernetes-dashboard
> dashboard-admin-token-?????? kubernetes.io/service-account-token 3 179d
$ kubectl describe secret dashboard-admin-token-?????? -n kubernetes-dashboard
Name:          dashboard-admin-token-??????
Namespace:     kubernetes-dashboard
Labels:        <none>
Annotations:   kubernetes.io/service-account.name: dashboard-admin
                kubernetes.io/service-account.uid: <UUID>
Type:          kubernetes.io/service-account-token
Data
====
ca.crt:      526 bytes
namespace:   20 bytes
token:       <super long string which is your plaintext bearer token>
```

- NOTE: you can do this because of your kube config file which gives you root

- **Start proxying:**

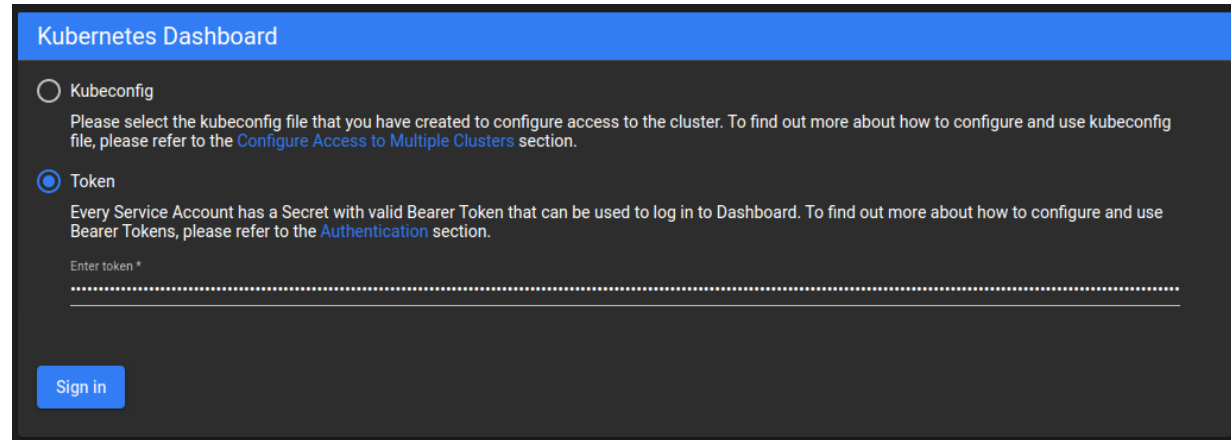
```
$ kubectl proxy # this is a blocking process and opens port 8001 on your localhost
```

- Note: you can make it so that the dashboard is always exposed too by editing the kubernetes-dashboard 'Service'
 - Kubectl edit service kubernetes-dashboard -n kubernetes-dashboard; make Type: ClusterIP → NodePort
 - Since we use Bearer Token, not advised



Step 4.c: Logging into your Dashboard

- `http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/pod?namespace=_all`
- Click “Token”
- Enter your Token
- Click Sign In



The screenshot shows the 'Kubernetes Dashboard' login page. It has a blue header bar with the text 'Kubernetes Dashboard'. Below the header, there are two radio button options: 'Kubeconfig' and 'Token'. The 'Token' option is selected. Under 'Kubeconfig', there is a text instruction: '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.' Under 'Token', there is a text instruction: '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.' Below the 'Token' instruction, there is a text input field with the placeholder 'Enter token *'. At the bottom left, there is a blue button labeled 'Sign in'.

Welcome to your cluster

The screenshot displays the Kubernetes dashboard interface. The left sidebar contains a navigation menu with categories like Cluster, Namespace, Overview, Workloads, Config and Storage, Custom Resource Definitions, and Settings. The main content area is divided into two sections: 'Cluster Roles' and 'Namespaces'.

Cluster Roles

Name	Age
system:certificates.k8s.io:legacy-unknown-approver	1 month
system:certificates.k8s.io:kube-apiserver-client-approver	1 month
system:certificates.k8s.io:kube-apiserver-client-kubelet-approver	1 month
system:controller.endpointlice-controller	1 month
system:certificates.k8s.io:kubelet-serving-approver	1 month
cert-manager:webhook:webhook-requester	5 months
cert-manager:controller:orders	5 months
cert-manager:controller:certificates	5 months
cert-manager:controller:challenges	5 months
cert-manager:controller:issuers	5 months

1 - 10 of 74

Namespaces

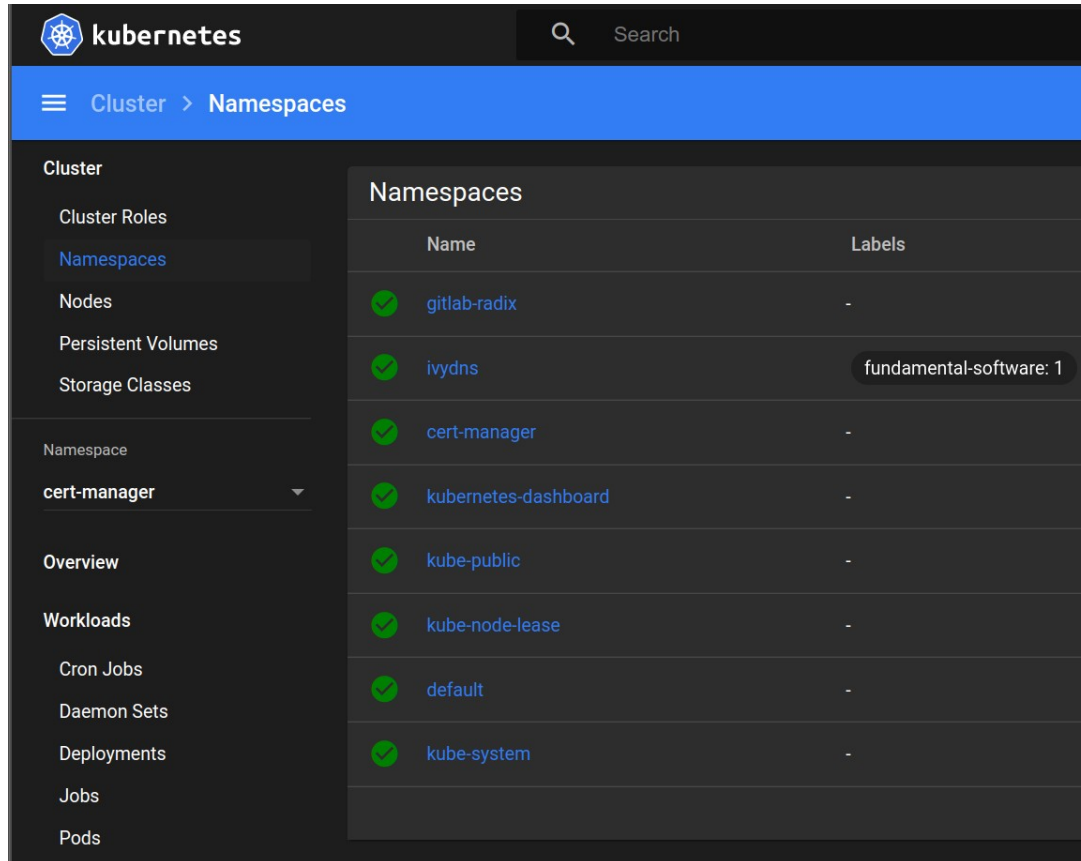
Name	Labels	Phase	Age
gitlab-radix	-	Active	4 months
hydra	fundamental-software: 1	Active	5 months
cert-manager	-	Active	5 months
kubernetes-dashboard	-	Active	5 months
kube-public	-	Active	5 months
kube-node-lease	-	Active	5 months
default	-	Active	5 months
kube-system	-	Active	5 months

1 - 8 of 8

Nodes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
------	--------	-------	----------------------	--------------------	-------------------------	-----------------------	-----

Cluster Namespaces

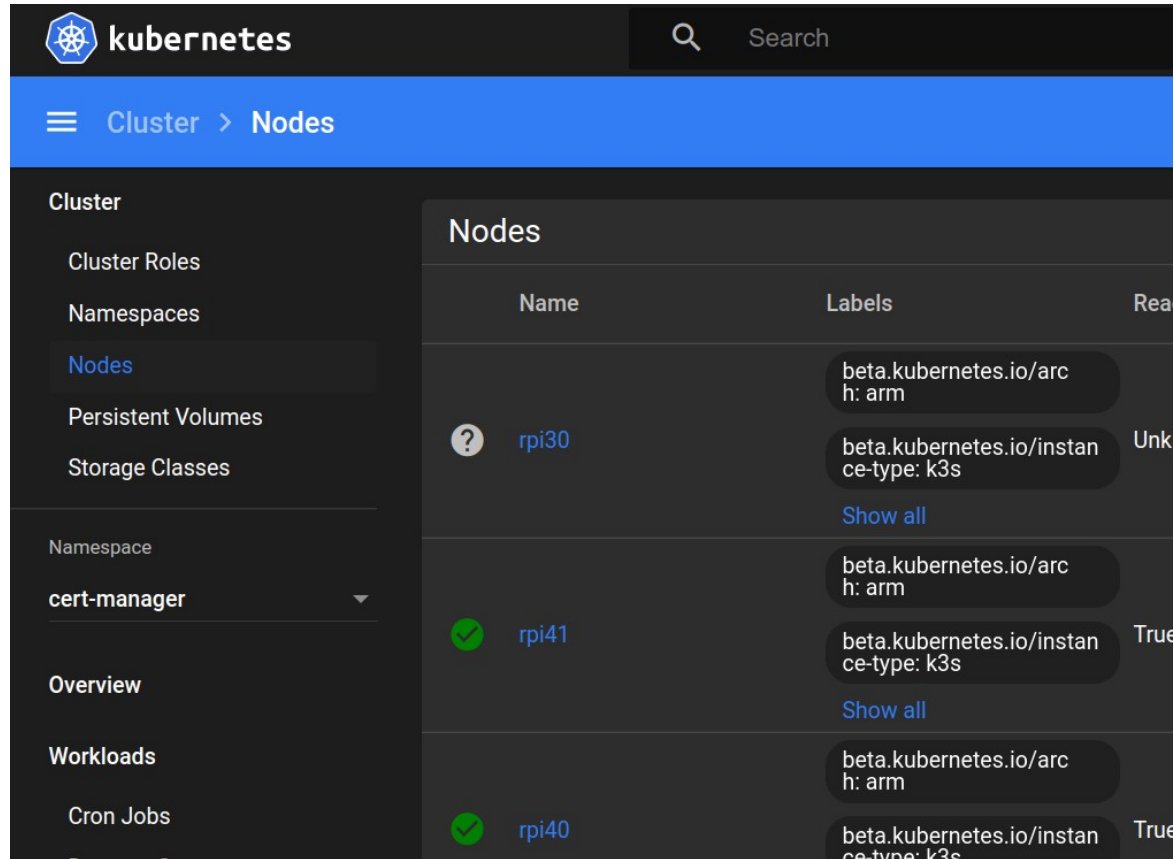


The screenshot shows the Kubernetes dashboard interface. At the top, there's a header with the Kubernetes logo and a search bar. Below the header, a blue navigation bar indicates the current path: Cluster > Namespaces. On the left side, there's a sidebar menu with various options: Cluster (selected), Cluster Roles, Namespaces (highlighted in blue), Nodes, Persistent Volumes, Storage Classes, Namespace (with a dropdown menu showing 'cert-manager'), Overview, Workloads (with sub-items: Cron Jobs, Daemon Sets, Deployments, Jobs, and Pods).

The main content area displays a table titled 'Namespaces'. The table has two columns: 'Name' and 'Labels'. Each row represents a namespace and includes a green checkmark icon in the first column. The namespaces listed are: gitlab-radix, ivydns (with a label 'fundamental-software: 1'), cert-manager, kubernetes-dashboard, kube-public, kube-node-lease, default, and kube-system.

Name	Labels
gitlab-radix	-
ivydns	fundamental-software: 1
cert-manager	-
kubernetes-dashboard	-
kube-public	-
kube-node-lease	-
default	-
kube-system	-

Cluster Nodes



kubernetes Search

Cluster > Nodes

Cluster

- Cluster Roles
- Namespaces
- Nodes**
- Persistent Volumes
- Storage Classes

Namespace: cert-manager

Overview

Workloads

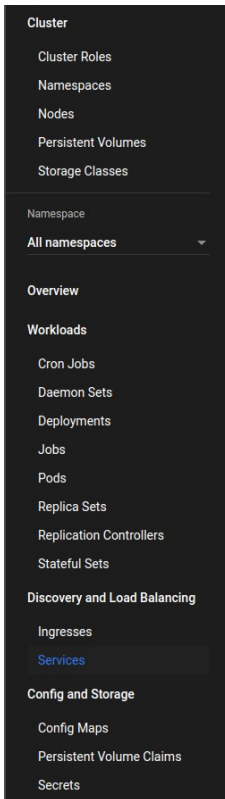
- Cron Jobs

Nodes

	Name	Labels	Ready
?	rpi30	<ul style="list-style-type: none">beta.kubernetes.io/arch: armbeta.kubernetes.io/instance-type: k3s Show all	Unknown
✓	rpi41	<ul style="list-style-type: none">beta.kubernetes.io/arch: armbeta.kubernetes.io/instance-type: k3s Show all	True
✓	rpi40	<ul style="list-style-type: none">beta.kubernetes.io/arch: armbeta.kubernetes.io/instance-type: k3s	True

Other Cluster Artifacts

- **Persistent Volumes & Storage Classes**
 - Provide storage, typically pretty tied to cloudy providers
 - Storage Class: think of this as “EBS”, “S3”, ...
 - Persistent Volumes: Think of this as “an EBS instance”
 - You can set up your cluster so that if something goes “I claim a persistent volume of class X”, that it automatically goes and provisions it with your cloud provider.



Namespaced Cluster Artifacts

- **Per namespace:**

- Cron Jobs: Cron job defined as and running in a container
- Daemon Sets
- Deployments & Pods
 - Pod: Set of containers that runs together on the same node (for whatever reason)
 - Deployment: One or more Pods that need to be deployed together (or make up a logical deployment unit)
- Jobs
- Replica Sets, Replication Controllers
- Stateful Sets: for if you are running something with a lot of 'state', e.g. an RDBMS... if you must
- Ingresses & Services: It's not because you have a container running that you can access said container
 - Ingress: provides a doorway through which you can enter the cluster to reach a service
 - Service: provides routing through to the actual pod & container that is running the thing you want to use
 - For instance:
 - I'd like to make it so that `http://cluster/foo` gets me my website running "in my cluster"
 - I have a container running inside a pod that is running apache
 - That pod can be on any one of my nodes, and can be moved from one node to another
 - How do I make it so that I have a constant entry-point to access what is basically running in that container?
 - The Ingress does the 'public part'; it is where I say "look /foo should be a thing and go to some place"; that 'some place' is the Service; so the ingress links a public path to a service
 - The Service does the backend work, figuring out where your container is running and then proxying through to it
 - So the Ingress does the front of the shop, hands it to the service, who then figures out what needs to happen in the back of the shop and makes sure that your request is routed to the actual node that is running your container
- Config Maps & Secrets: mount configuration information and secrets as files into your containers, as defined in your YAML files
- Persistent Volume Claims: Claims of persistent volumes

Cluster
Cluster Roles
Namespaces
Nodes
Persistent Volumes
Storage Classes
Namespace
All namespaces
Overview
Workloads
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Discovery and Load Balancing
Ingresses
Services
Config and Storage
Config Maps
Persistent Volume Claims
Secrets



Kube-system namespace

kubernetes Search +

Overview

Cluster

- Cluster Roles
- Namespaces
- Nodes
- Persistent Volumes
- Storage Classes


Namespace

kube-system

Overview

Workloads

Workload Status



Daemon Sets Deployments Jobs Pods Replica Sets

Daemon Sets

Name	Labels	Pods	Age ↑	Images
svcib-traefik	objectset.rio.cattle.io/hash: f31475152bf70655d3c01ed368e90118938f6ea svccontroller.k3s.cattle.io/node-selector: false	3 / 2	5 months	rancher/klipper-lb:v0.1.2 rancher/klipper-lb:v0.1.2

1 - 1 of 1 |< < > >|

Deployments

Name	Labels	Pods	Age ↑	Images
traefik	app: traefik chart: traefik-1.81.0	1 / 1	5 months	rancher/library-traefik:1.7.19
metrics-server	k8s-app: metrics-server objectset.rio.cattle.io/hash: e10e245e13e46a725c9ddddd4f9eb239f147774fd	1 / 1	5 months	rancher/metrics-server:v0.3.6

kubernetes Search +

Overview

Cluster

- Cluster Roles
- Namespaces
- Nodes
- Persistent Volumes
- Storage Classes


Namespace

kubernetes-dashboard

Overview

Workloads

Workload Status



Deployments Pods Replica Sets

Deployments

Name	Labels	Pods	Age ↑	Images
dashboard-metrics-scraper	k8s-app: dashboard-metrics-scraper	1 / 1	5 months	kubernetesui/metrics-scraper:v1.0.1
kubernetes-dashboard	k8s-app: kubernetes-dashboard	1 / 1	5 months	kubernetesui/dashboard:v2.0.0-beta8

1 - 2 of 2 |< < > >|

Pods

Name	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Age ↑
dashboard-metrics-scraper-76585494d8-dvx9c	k8s-app: dashboard-metrics-scraper pod-template-hash: 76585494d8	rpi40	Running	11	-	-	5 months
kubernetes-dashboard-5996555fd8-mpd4j	k8s-app: kubernetes-dashboard pod-template-hash: 5996555fd8	rpi40	Running	12	-	-	5 months

Step 5: Installing more things in your cluster

- **Typically: use Helm unless it's software you built yourself**
- **“Helm Charts” are templates for full applications**
 - All moving components in one go
 - You “just fill in some variables” through a values file that you specify when deploying it
- **Add a repo (e.g. jetstack.io)**
 - `$ helm repo add jetstack https://charts.jetstack.io`
 - <https://hub.helm.sh/charts/jetstack>
 - `$ helm repo update`
 - `$ nano helmchart_values_file.yaml`
 - `$ helm install/upgrade <package>`
 - cert-manager: Automate issuance of TLS Certificates, be your own CA
 - gitlab-runner: modified from official docs
 - traefik: application router
 - Example
 - `$ helm upgrade --install --atomic --kubeconfig /etc/rancher/k3s/k3s.yaml --namespace cert-manager --version v0.15.2 --set webhook.enabled=true cert-manager jetstack/cert-manager --values /etc/kubernetes/helm-values/gitlab.values.yaml`
- **Outside the scope of this presentation**



My set-up is heavy on Ansible

- **Through Ansible, I upload definitions to the main node**
 - /etc/kubernetes/ (chown -R root:kubernetes)
 - ./configs/: things I define myself to run in the cluster
 - ./helm-values/: values for helm-charts I'll install
- **Through Ansible, I then apply what is there from the main node**
 - \$ kubectl apply -f /etc/kubernetes/config/*.yaml
 - \$ helm upgrade --install <etc> --values
"/etc/kubernetes/helm-values/stuff.yaml"
- **Obviously, all of this is checked into source code control**
- **\$ ansible-playbook -i inventory.yaml playbook.yaml**
 - Using a password vault of course...



What am I running in there?

- **Kubernetes addons run within kubernetes**

- Traefik: Application Routing/Service Load Balancer (i.e. :443/ivydns-report goes to the IvyDNS report application)
- Metrics-server: Metric collection from nodes
- Dashboard/WebUI with metrics scraper
- Local-path-provisioner
- CoreDNS: everything inside your cluster gets its own internal domain (and every 'pod' has its own IP as well, btw)
- Cert-manager: so I can use HTTPS on services I put in there

- **IvyDNS stuff**

- Domain Classifiers & Collectors
- IvyDNS report generation as a containerized cron job
- IvyDNS report serving over https

- **GitLab Runner for a side-project**



Running other stuff in there

- **If there's a pullable container for it, you can run it in Kubernetes**
 - Likely some work involved in defining it in k8s YAML
- **Including things you build yourself**
 - Push them into docker hub (or wherever)



Running your own stuff in your Cluster

- **Raspberry Pi's are ARM**
- **What you run on them, must be ARM**
 - And not all containers have an ARM build... :(
 - Most do though (e.g. python3, apache, node, ...)
- **Docker buildx to build ARM on x86_64**
 - Enable experimental mode for this
 - `$ docker buildx build --platform linux/arm/v7 -t username/project:tag .`
 - <https://docs.docker.com/buildx/working-with-buildx/>
- **You *can* build multiple architectures for a single container definition**



Storage in your cluster

- **I use local-path provisioning**
 - A local path on the node backs the storage
 - Which means that any node must have this path available
 - Things stored there are stored 'on the local node'
 - But, that local path is actually an NFS Mount which is the same on each node:
 - /mnt/clusterfs
 - Granted, this is a bit of a hack
 - If it's stupid but it works, it's not stupid (for now)
 - The local path is just partitioned up:
 - /mnt/clusterfs/ivydns-report
 - /mnt/clusterfs/gitlab-runner
 - Etc.
- **I'm working on something more sustainable for this**



Summary

- **What and why is kubernetes**
- **Raspberry Pi's are totally cool**
- **Deploying k3s kubernetes on Raspberry Pis**
- **Setting up the WebUI as your first deployment**
 - Having a look at the different things in the WebUI
- **Deploy more things in your Cluster**
 - kubectl and helm
- **Some notes about deploying your own stuff in your Cluster**
 - A note on ARM and container architectures
 - A note on Storage



Links

- <https://k3s.io/>
- <https://www.kubernetes.io>
 - Documentation is pretty good with interactive environment
 - <https://kubernetes.io/docs/home/>
 - <https://kubernetes.io/docs/tutorials/kubernetes-basics/>
- <https://helm.sh/>

