Containers and Orchestration Oh My!

Frank Seesink, UNC Chapel Hill
First, a message from our sponsor...
What I picture in my head...
What it ends up looking like...

Actually that’s not quite right. The guy who made this is **clearly** more talented.
Who am I?

Frank Seesink

- Senior Network Engineer, UNC Chapel Hill
- Part of network DevOps group
- Involved in network automation for years
- JOAT - databases, OSes, networking,...
- Using Docker since ~2014(?)
- Red Hat Open Shift, Rancher Desktop, K3s, Helm, Kompose, Lens
- Working towards GitOps
Story Time...
Current Setup

- django
- Red Hat
- OpenShift
- SQLite
- db
- Nornir
- gunicorn

Work environment
Current Setup

GitLab on-prem

Webhook

Red Hat OpenShift Container Platform
Current Setup

https://xkcd.com/2347
The Plan
Out-of-Band Server

GitLab

- GitLab Runners
- CI/CD pipelines

Server

- Webserver (NGINX)
- Django/Gunicorn
- PostgreSQL
- Celery
- Redis

• CI/CD pipelines
That’s the backstory.
Now a flashback...
In the beginning... you had a computer...
That computer ran an OS...
Then came Virtual Machines (VMs)...

https://www.vmware.com/

https://www.virtualbox.org/

https://www.proxmox.com

https://www.parallels.com/
Then came Virtual Machines (VMs)…
Now what about containers?
Container Terms

container: A container is a running process with resource and capability constraints managed by a computer’s operating system. The files available to the container process are packaged as a container image. Containers run adjacent to each other on the same machine, but typically the operating system prevents the separate container processes from interacting with each other.

— https://glossary.cncf.io/container/
Container Terms

**image**: a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

**ephemeral**: lasting for a very short time.

“Containers are ephemeral.”
“Containers should be ephemeral.”
Short History

Installation time
- hours/days
- minutes

Startup time
- hours/days
- minutes
- seconds
Container History

August 2008

March 2013

September 2014

December 2014
Container History

August 2008

March 2013

September 2014

December 2014
Container History
Container History

donated image format
Docker

To install on Linux:
• yum/dnf
• apt-get

Docker Desktop
• for Linux
• for macOS
• for Windows

https://www.docker.com/products/docker-desktop/
Docker

Docker Desktop

- for Linux
- for macOS
- for Windows
Docker

What does it do?
- CLI driven
- Download/build/upload images
- Run/manage containers built from images
- Combine containers into apps using Docker Compose
- Run Kubernetes!
  (disabled by default)

Docker Desktop
- for Linux
- for macOS
- for Windows
Docker

How does it do it?
• Dockerd* daemon
• 'docker' CLI
  - `docker run -it ubuntu:22.04 /bin/bash`
  - `docker ps -a`
  - `docker volume ls`
  - `docker build ...`
• Dockerfiles (YAML) to create images
  • Layers
    e.g., alpine:latest + app
• Docker Compose files (YAML) to create multi-container apps

Docker Desktop
• for Linux
• for macOS
• for Windows
Docker Desktop
- for Linux
- for macOS
- for Windows

Docker Hub (hub.docker.com)

Container Registry

Docker Hub

dockerd (daemon)

docker (CLI)

redis
celery
db
MyApp

image
Dockerfile

# Use official Python image from hub.docker.com
# If you have M1 Mac, use
# FROM: arm64v8/python
FROM ${DOCKER_ARCH}python

# NOTE: To fully mimic production, we can set tags to match specific versions
# of things like Python.
# Also be aware that trying to use python:3-alpine will fail due to the
# need for additional packages for LDAP/etc. Hence using stock image.
# Install needed modules so LDAP bits in pip modules will install properly
RUN apt-get update
RUN apt-get -y install libsasl2-dev python-dev libldap2-dev libssl-dev
RUN cd /

# Mount host directory where docker-compose.yml is located, which should be
# root of MyApp project source code so it appears as /MyApp in the container.
# This exposes things like the requirements.txt file so we can install all the
# relevant Python modules for this project
RUN --mount=type=bind,source=.,target=/MyApp pip install -r /MyApp/requirements.txt
RUN pip install gunicorn
RUN cd /MyApp

CMD ["echo", "MyApp is running..."]
Docker

Docker Desktop
- for Linux
- for macOS
- for Windows

hub.docker.com

Container Registry

dockerd (daemon)
docker (CLI)

image

redis
 celery
 db
 MyApp
Docker

Docker Desktop
• for Linux
• for macOS
• for Windows

dockerd (daemon)

redis
celery
db
MyApp
Docker

Docker Desktop
- for Linux
- for macOS
- for Windows

dockerd (daemon)

redis

celery

db

MyApp

Networking

Persistent Storage

Application
Docker Compose

docker-compose.yml

Docker Desktop
- for Linux
- for macOS
- for Windows
Docker Limitations

What are Docker’s limitations?
• Dockerd* daemon runs as root by default
• Docker Engine is open source (Apache License v2). Docker Desktop is NOT. Nor is it free for all use.

"Commercial use of Docker Desktop at a company of more than 250 employees OR more than $10 million in annual revenue requires a paid subscription (Pro, Team, or Business)."
– https://www.docker.com/pricing/

"4.2 Specific License Limitations – Docker Desktop.
(a) The Docker Desktop component of the Service at the level of the Personal Offering (as described on the Pricing Page) is further restricted to: (i) your “Personal Use”, (ii) your “Educational Use”, (iii) your use for a non-commercial open source project, and (iv) your use in a “Small Business Environment”.
(b) For purposes of this Section 4.2: (i) “Personal Use” is the use by an individual developer for personal use to develop free or paid applications, (ii) “Educational Use” is the use by members of an educational organization in a classroom learning environment for academic or research purposes or contribution to an open source project..."
– https://www.docker.com/legal/docker-subscription-service-agreement/
Docker Alternatives

What are the alternatives?

- **Colima** - container runtimes on macOS (and Linux) [CLI]
- **Red Hat OpenShift Local** (formerly Red Hat CodeReady Containers)
- **Podman**
- **Podman Desktop**

**Docker Desktop**
- for Linux
- for macOS
- for Windows
Colima - container runtimes on macOS (and Linux) with minimal setup.

Features
- Intel and M1 Macs support
- Simple CLI interface
- Docker and Containerd support
- Port Forwarding
- Volume mounts
- Kubernetes

https://github.com/abiosoft/colima
Red Hat OpenShift Local

**Hardware Requirements:**
Red Hat OpenShift Local is supported on AMD64, Intel 64, and Apple M1 processor architectures.

For OpenShift Container Platform:
- 4 physical CPU cores
- 9 GB of free memory
- 35 GB of storage space

For the Podman container runtime:
- 2 physical CPU cores
- 2 GB of free memory
- 35 GB of storage space

**OS Requirements:**
- Windows 10 Fall Creators Update (version 1709) or later
- macOS 11 Big Sur or later
- only on the latest two Red Hat Enterprise Linux/CentOS 8 and 9 minor releases and on the latest two stable Fedora releases


https://developers.redhat.com/products/openshift-local/overview
"Podman is a daemonless container engine for developing, managing, and running OCI Containers on your Linux System. Containers can either be run as root or in rootless mode."

https://podman.io/
Podman Desktop

https://podman-desktop.io/
Orchestration

"In system administration, orchestration is the automated configuration, coordination, and management of computer systems and software."

Examples of Container Orchestration Tools

- Kubernetes
- Docker Swarm
- Apache Mesos
- HashiCorp Nomad
Kubernetes (K8s)

Cluster
  • Control Plane
    • API
    • Etcd
      • distributed key/value store

  • Nodes (workers)
    • Node1
      • Kubelet1
        • Pod1 ("Application 1")
        • Container1
        • Container2
      • Pod2 ("Application 2")
      • Pod3 ("Application 3")
    • Node2
      • Kubelet2
        • Pod1
        • Pod2
    • Node3
      • Kubelet3
        • Pod1
        • ...

"Containers on steroids"

"Cube cuddle"
Kubernetes (K8s)

"Containers on steroids"

A lot of moving parts
Kubernetes (K8s)

containerd
cri-o
etcd
CoreDNS
HELM
Kubernetes (K8s)

- containerd
- cri-o
- etcd
- CoreDNS
- HELM
Kubernetes (K8s)

"The package manager for Kubernetes"
— https://helm.sh/

- Docker Compose file
- Dev tool
- Difference:
  - Mount a local volume

- Helm Chart
- Deployment tool
- Difference:
  - No local volumes

https://kompose.io/
<table>
<thead>
<tr>
<th>Environment</th>
<th>Platforms</th>
<th>Status</th>
<th>Tag</th>
<th>Volumes</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker</td>
<td>2 stacks, 4 containers</td>
<td>up</td>
<td>Tag-04</td>
<td>2 volumes</td>
<td>3 images</td>
</tr>
<tr>
<td>Edge</td>
<td>2 stacks, 4 containers</td>
<td>up</td>
<td>Tag-04</td>
<td>2 volumes</td>
<td>3 images</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>6 stacks, 3 containers</td>
<td>up</td>
<td>Tag-03</td>
<td>7 volumes</td>
<td>4 images</td>
</tr>
<tr>
<td>Nomad</td>
<td>5 stacks, 4 containers</td>
<td>down</td>
<td>Tag-04</td>
<td>2 volumes</td>
<td>2 images</td>
</tr>
<tr>
<td>Swarm</td>
<td>18 stacks, 200 containers</td>
<td>heated</td>
<td>Tag-03</td>
<td>22 volumes</td>
<td>18 images</td>
</tr>
<tr>
<td>Azure</td>
<td>14 stacks, 17 containers</td>
<td>up</td>
<td>Tag-05</td>
<td>5 volumes</td>
<td>12 images</td>
</tr>
</tbody>
</table>

- **Web-based**
- **Can run in container**
- Desktop app (Linux, macOS, Windows)
- Provides similar experience to Kubernetes Dashboard
- Can connect to any Kubernetes cluster via API using kubeconfig* file

* typically ~/.kube/config
• a drop-in replacement for **docker** CLI
• You can do all your workloads using containerd
End Flashback
Let’s return to our show in progress...
So How to Build Our Out-of-Band Server?
OPTION #1: Build from Source

ADVANTAGES:

• Simplest to understand
• Easiest to do initial build

DISADVANTAGES:

• No repeatability*; more difficult to maintain
• Long-term will take most time to maintain
• No rollback/recovery if a build goes bad
• More downtime
OPTION #2: Docker-ize

ADVANTAGES:

• More repeatable with Dockerfiles and docker-compose
• Allows for rollback/recovery
• Relatively easy to spin up/down manually

DISADVANTAGES:

• Still doesn’t automate code updates*
• A dev tool/Security concerns (e.g., Docker daemon runs as root)
• Podman alternative has other issues

*GitLab CI/CD may help with this
OPTION #3: Single-node K3s Cluster

ADVANTAGES:

- Offers full Kubernetes experience (with all benefits that brings) with less sysadmin overhead ("Tastes great; less filling")
- Very robust / scalable / repeatable
- Easiest to maintain long-term

DISADVANTAGES:

- Most complex to initially understand / setup.

https://k3s.io/
Lightweight Kubernetes

The certified Kubernetes distribution built for IoT & Edge computing

This won’t take long...

curl -sfL https://get.k3s.io | sh -
# Check for Ready node, takes ~30 seconds
sudo k3s kubectl get node

For detailed installation refer to the docs

https://k3s.io/
K3s Installation

This is actually easy. Key preparations:

1. Setup FQDN for the server
   (IMPORTANT: K3s uses its own resolver)

2. Configure host firewall for proper communication (80, 443, 6443/TCP; etc.)

3. Use K3s Quick-Start Guide step to install:

   `sudo curl -sfL https://get.k3s.io | sh -`
K3s Installation

Additional steps:

4. Setup local users to have cluster access

5. Install Helm (https://helm.sh/)

6. Configure TLS certificates for Traefik
K3s Maintenance

Step #1:

```
sudo curl -sfL https://get.k3s.io | sh -
```

Step #2:

There is no step #2.

Everything else is like any other Linux server, so this server can be managed via Ansible, etc. like any other.
Current Setup

Kubernetes (K8s) =

Red Hat OpenShift
Container Platform

Kubernetes is
Kubernetes is
Kubernetes

+
But what does this really mean?
What does Kubernetes even DO for us?
Traditional App Deployment

• Build a server (e.g., a "LAMP" stack with Linux, Apache, MySQL, Perl/PHP/Python)

• Install application code

• Configure all the bits ("artisanal")

• Ignore scaling issues, 10K problem, etc.
Full-blown Kubernetes (K8s) Architecture

Normally min. 4 nodes

Raft Consensus Algorithm == odd # nodes
K3s - Single Node Cluster

Minimum can be a single node for everything
K8s Default Setup

Namespace

kube-system

Ingress controller

kube-dns

metrics-server

default

Kubernetes API

kube-public

kube-node-lease

All pods assigned 10.43.x.x

80/TCP

443/TCP

53/TCP

9153/TCP

6443/TCP

443/TCP

443/TCP
K3s Default Setup

Namespace

kube-system

80/TCP
443/TCP

Traefik

CoreDNS

53/TCP
9153/TCP

metrics-server

443/TCP

default

443/TCP

Kubernetes API

All K3s pods assigned 10.43.x.x

6443/TCP
K3s Default Setup

Namespace
kube-system

Traefik
80/TCP
443/TCP

Kubernetes API
443/TCP

6443/TCP

All K3s pods assigned 10.43.x.x
K3s Default Setup

Namespace
- kube-system
  - Traefik
    - 80/TCP
    - 443/TCP
- default
  - All K3s pods assigned 10.43.x.x
  - All application pods/containers assigned 10.42.x.x

pod (think "application")
- container
  - Django/gunicorn
    - 8000/TCP
- container
  - PostgreSQL
    - 5432/TCP
- container
  - Celery (or Redis,...)
    - 5672/TCP
K3s Default Setup

Namespace
- **kube-system**
- **default**

**pod (think "application")**
- **container** Django/gunicorn
  - 80/TCP
  - 8000/TCP
- **container** PostgreSQL
  - 5432/TCP
- **container** Celery (or Redis,...)
  - 5672/TCP

**Traefik**
- 80/TCP
- 443/TCP

All K3s pods assigned 10.43.x.x
All application pods/containers assigned 10.42.x.x

6443/TCP

K3S
K3s Default Setup

Namespace

kube-system

default

pod (think "application")

container Django/gunicorn

80/TCP

container PostgreSQL

5432/TCP

container Celery (or Redis, ...)

5672/TCP

Internet

appFQDN:80

Traefik

All K3s pods assigned 10.43.x.x

6443/TCP

All application pods/containers assigned 10.42.x.x

80/TCP

5432/TCP

5672/TCP
NOTES

- Deployments are configured via the Kubernetes API
- Services are configured via the Kubernetes API
- **Traefik** monitors changes in the cluster (such as service ports being exposed) via the Kubernetes API
- ...

...
KEY: EVERYTHING communicates via the Kubernetes API
Kubectl

**kubectl** is the Command Line Interface (CLI) tool that connects to the Kubernetes API (6443/TCP). e.g.,

```
kubectl cluster-info
```
```
kubectl api-resources
```
```
kubectl get [all|nodes|pods|services|endpoints|namespaces/ns] [-A]
```
Kubectl

Red Hat OpenShift has a tool called simply “oc” (“OpenShift CLI”). This is simply OpenShift’s kubectl.
Kubectl

Now if you have multiple Kubernetes clusters you manage/interface with (e.g., OpenShift, K3s, etc.), how do `kubectl` and other apps know which API to connect to?
apiVersion: v1
clusters:
- cluster:
  server: https://api.cloudapps.unc.edu:443
  name: api-cloudapps-unc-edu:443
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZ...
  server: https://192.168.125.10:6443
  name: default
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZ...
  server: https://127.0.0.1:6443
  name: rancher-desktop
contexts:
- context:
  cluster: default
  user: default
  name: default
- context:
  cluster: api-cloudapps-unc-edu:443
  namespace: dept-its-routerproxy
  user: fseesink/api-cloudapps-unc-edu:443
  name: dept-its-routerproxy/api-cloudapps-unc-edu:443/fseesink
- context:
  cluster: rancher-desktop
  user: rancher-desktop
  name: rancher-desktop
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZ...
    client-key-data: LS0tLS1CRUdJTiBDRVJUSUZ...

- name: fseesink/api-cloudapps-unc-edu:443
  user:
    token: sha256~lR6K25T0CsowoJNtKEG03Fshy-9C028vcMZxmFwKP0
- name: rancher-desktop
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZ...
    client-key-data: LS0tLS1CRUdJTiBFOyB0UK1WQVRFIEt...
• Desktop app (Linux, macOS, Windows)
• Provides similar experience to Kubernetes Dashboard
• Can connect to any Kubernetes cluster via API using kubeconfig* file

* typically ~/.kube/config
Where do we get Kubeconfig?

On Server: /etc/rancher/k3s/k3s.yaml

```
apiVersion: v1
clusters:
  - cluster:
      certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZ...
      server: https://127.0.0.1:6443
      name: default

current-context: default
current-context: default
current-context: default
kind: Config
preferences: {}
users:
  - name: default
    user:
      client-certificate-data: WWVhaCBJIGRvbid0IHRoaW5...
      client-key-data: WW91IHJlYWxseSB0aGluyBWQVRFIEt.
```

1. Copy this file to your local machine (~/.kube/config)
2. Replace IP with K3s server IP/FQDN
So How are Kubernetes Clusters Managed?
Application Manifests

```shell
kubectl apply -f <manifest>.yaml
kubectl delete -f <manifest>.yaml
```

- Namespace (optional)
- Deployment
- Service
- Ingress
Application Manifests

Namespace (optional)

```
# CREATE A NAMESPACE FOR THIS APPLICATION (OPTIONAL)
#
# NOTE: If you DO this, be sure to uncomment both the lines below
# and relevant `namespace:` attributes in other sections.
#---
# apiVersion: v1
# kind: Namespace
# metadata:
#   name: sample-appspace
```
# CONFIGURE DEPLOYMENT OF THIS APPLICATION
# This is where you define the containers which make up your app
# and specify what ports each container exposes

apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  # namespace: sample-appspace
  # Name of the namespace (optional; see above)
  labels:
    app: sample-app
spec:
  selector:
    matchLabels:
      app: sample-app
  replicas: 1
  # Number of replicas
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      # Containers are the individual pieces of your application that you want
to run. Sample uses NGINX container for testing.
      - name: nginx
        # Name of the container
        image: nginx:latest
        # The image you want to run
        ports:
          # Ports are the ports that your application uses.
          - containerPort: 80
            # The port that your application uses
---

# CONFIGURE SERVICE
#
# This is where you expose your container ports to the rest of the cluster

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sample-app
# namespace: sample-appspace
spec:
  selector:
    app: sample-app
# ---
  type: ClusterIP
# ClusterIP means this service can be accessed by any pod in the cluster
  ports:
    - name: http
      port: 80
```
Application Manifests

Ingress

---

# CONFIGURE INGRESS

# This is where you configure the external ingress route to your
# application.

# NOTE: Traefik should detect this automatically and stitch a path
# so that inbound traffic destined for this application's
# FQDN/path goes to this application.

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
  name: sample-app
  # namespace: sample-appspace
  # annotations:
  # ...

spec:
  rules:
    - host: "FQDN.of.sample-app"
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: sample-app
                port:
                  number: 80
Manifests

- You can have your namespace, deployment, service, ingress, etc. YAML files all separate, or...

- You can stitch all the separate YAML files of a pod together into a single file, with each segment separated by the usual 3 dashes. So you could use this approach to have one YAML file per application deployed.
Managing K3s from the CLI
Basic Kubectl Commands

$ kubectl cluster-info
Kubernetes control plane 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:https/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

$ kubectl api-resources

<table>
<thead>
<tr>
<th>NAME</th>
<th>SHORTNAMES</th>
<th>APIVERSION</th>
<th>NAMESPACED</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindings</td>
<td></td>
<td>v1</td>
<td>true</td>
<td>Binding</td>
</tr>
<tr>
<td>componentstatuses</td>
<td>cs</td>
<td>v1</td>
<td>false</td>
<td>ComponentStatus</td>
</tr>
<tr>
<td>configmaps</td>
<td>cm</td>
<td>v1</td>
<td>true</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>endpoints</td>
<td>ep</td>
<td>v1</td>
<td>true</td>
<td>Endpoints</td>
</tr>
<tr>
<td>tlsstores</td>
<td></td>
<td></td>
<td>true</td>
<td>TLSStore</td>
</tr>
<tr>
<td>traefikservices</td>
<td></td>
<td>traefik.containo.us/v1alpha1</td>
<td>true</td>
<td>TraefikService</td>
</tr>
</tbody>
</table>

$ kubectl get all

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>service/kubernetes</td>
<td>ClusterIP</td>
<td>10.43.0.1</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>40d</td>
</tr>
</tbody>
</table>

$ kubectl get nodes

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverbox01.net.unc.edu</td>
<td>Ready</td>
<td>control-plane,master</td>
<td>40d</td>
<td>v1.26.3+k3s1</td>
</tr>
</tbody>
</table>

$ kubectl get pods

No resources found in default namespace.

$ kubectl get services

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubernetes</td>
<td>ClusterIP</td>
<td>10.43.0.1</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>40d</td>
</tr>
</tbody>
</table>

$ kubectl get endpoints

<table>
<thead>
<tr>
<th>NAME</th>
<th>ENDPOINTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubernetes</td>
<td>192.168.125.10:6443</td>
<td>40d</td>
</tr>
</tbody>
</table>

$ kubectl get namespaces

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>Active</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system</td>
<td>Active</td>
<td>40d</td>
</tr>
<tr>
<td>kube-public</td>
<td>Active</td>
<td>40d</td>
</tr>
<tr>
<td>kube-node-lease</td>
<td>Active</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent</td>
<td>Active</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-runner</td>
<td>Active</td>
<td>40d</td>
</tr>
</tbody>
</table>

Why so little output? This is only the 'default' namespace.
```bash
kubectl get all -A
```

### Output Explained

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system pod/helm-install-traefik-crd-rk8vd</td>
<td>0/1</td>
<td>Completed</td>
<td>0</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system pod/helm-install-traefik-q9kq</td>
<td>0/1</td>
<td>Completed</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system pod/svclb-traefik-d9306a27-kqk4</td>
<td>2/2</td>
<td>Running</td>
<td>22 (4h36m ago)</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system pod/coredns-7c444649cb-72kvt</td>
<td>1/1</td>
<td>Running</td>
<td>5 (4h36m ago)</td>
<td>11d</td>
</tr>
<tr>
<td>kube-system pod/local-path-provisioner-79f67d76f8-v84z1</td>
<td>1/1</td>
<td>Running</td>
<td>5 (4h36m ago)</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system pod/traefik-66c46d954f-5qhrq</td>
<td>1/1</td>
<td>Running</td>
<td>22 (4h36m ago)</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent pod/k3s-agent-gitlab-agent-c7d5b5cfd-lbcns</td>
<td>1/1</td>
<td>Running</td>
<td>4 (4h36m ago)</td>
<td>5d4h</td>
</tr>
<tr>
<td>kube-system pod/metrics-server-5f9f776df5-ssm4j</td>
<td>1/1</td>
<td>Running</td>
<td>23 (4h36m ago)</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-runner pod/gitlab-runner-6db665f9dc-89m7x7</td>
<td>1/1</td>
<td>Running</td>
<td>22 (4h36m ago)</td>
<td>40d</td>
</tr>
</tbody>
</table>

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default service/kubernetes</td>
<td>ClusterIP</td>
<td>10.43.0.1</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system service/kube-dns</td>
<td>ClusterIP</td>
<td>10.43.0.10</td>
<td>&lt;none&gt;</td>
<td>53/UDP,53/TCP,9153/TCP</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system service/metrics-server</td>
<td>ClusterIP</td>
<td>10.43.47.30</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system service/traefik</td>
<td>LoadBalancer</td>
<td>10.43.63.81</td>
<td>192.168.125.10</td>
<td>80:31967/TCP,443:30819/TCP</td>
<td>40d</td>
</tr>
</tbody>
</table>

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>NODE SELECTOR</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubernetes daemonset.apps/svclb-traefik-d9306a27</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>&lt;none&gt;</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/k3s-agent-gitlab-agent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>&lt;none&gt;</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/metrics-server</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>&lt;none&gt;</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/gitlab-runner</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>&lt;none&gt;</td>
<td>40d</td>
</tr>
</tbody>
</table>

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system deployment.apps/traefik</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system deployment.apps/coredns</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system deployment.apps/local-path-provisioner</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/k3s-agent-gitlab-agent</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/metrics-server</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent deployment.apps/gitlab-runner</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
</tbody>
</table>

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system replicaset.apps/traefik-66c46d954f</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system replicaset.apps/coredns-7c444649cb</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent replicaset.apps/k3s-agent-gitlab-agent-76fc86cb9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent replicaset.apps/k3s-agent-gitlab-agent-55fbfb5b9c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5d4h</td>
</tr>
<tr>
<td>kube-system replicaset.apps/local-path-provisioner-79f67d76f8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent replicaset.apps/k3s-agent-gitlab-agent-c7d5b5cfd</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5d4h</td>
</tr>
<tr>
<td>kube-system replicaset.apps/metrics-server-5f9f776df5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
<tr>
<td>gitlab-agent-k3s-agent replicaset.apps/gitlab-runner-6db665f9dc</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40d</td>
</tr>
</tbody>
</table>

#### NAMESPACE

<table>
<thead>
<tr>
<th>NAME</th>
<th>COMPLETIONS</th>
<th>DURATION</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system job.batch/helm-install-traefik-crd</td>
<td>1/1</td>
<td>13s</td>
<td>40d</td>
</tr>
<tr>
<td>kube-system job.batch/helm-install-traefik</td>
<td>1/1</td>
<td>15s</td>
<td>40d</td>
</tr>
</tbody>
</table>
• Normally Traefik uses LetsEncrypt to add SSL certificates to apps. This, however, requires that the LetsEncrypt servers can reach into the Kubernetes cluster to verify things on the website.

• This will not work in some environments, notably ones for internal use only.
Traefik Default SSL Cert.

```yaml
---
apiversion: traefik.containo.us/v1alpha1
kind: TLSStore
metadata:
  name: default
  namespace: kube-system
spec:
  defaultCertificate:
    secretName: default-certificate
---
apiVersion: v1
kind: Secret
metadata:
  name: default-certificate
  namespace: kube-system
  type: kubernetes.io/tls
data:
  # To create the following 2 lines, take .pem/.key files and base64 them; e.g.,
  # cat FQDN.of.sample-app.pem | base64
  # cat FQDN.of.sample-app.key | base64
tls.crt: LS0tLS1CRUdJTiBD RVJUSUZJQ0FURS0tLS0tCk1JSUd3VENDQmFtZ0F3SUJBZ0lRZHNFRE...
tls.key: LS0tLS1CRUdJTiBQQUk\WQVRFIEtFW S0tLS0tCk1JSUV2Z0lCQURBTkJna3 Foa2lH0XcwQk...
```
"CNCF is the open source, vendor-neutral hub of cloud native computing, hosting projects like Kubernetes and Prometheus to make cloud native universal and sustainable."

— https://www.cncf.io/
Thank You

https://frank.seesink.com/presentations/Internet2TechEx-Fall2023/

Frank Seesink
frank@seesink.com
frank@unc.edu