Learn Practical Kubernetes

By

Stanley Stephen

Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/
Email: s.stanley.mca@gmail.com

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Practical - Kubernetes

This Practical - Kubernetes covers Introduction, Setting up your own Kubernetes in AWS VM, Kubernetes Resources, Deploying sample applications, HELM, Monitoring the Kubernetes with Prometheus, Grafana, Alert Manager.

Some contents (IMAGE/Text) are copied from freely available resources from internet / Open source materials. Thanks to the original authors.

This practical guide is free to use.

---

1. Kubernetes installation (using MiniKube)

VM details

AWS VM (Minimum):

- 2vCPU/4GB RAM/20GB HDD
- Ubuntu 18.04

AWS VM (Recommended):

- 4vCPU/8GB RAM/50GB HDD
- Ubuntu 18.04

In the security groups allow the incoming traffic for the TCP port 80 and 8080.

Tools Installation

docker

```
sudo apt-get update

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

sudo apt update

sudo apt install -y docker-ce

sudo service docker status

sudo usermod -aG docker $USER && newgrp docker
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

conntrack & socat

```
sudo apt-get install -y conntrack socat
```

kubectl

```
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

sudo mv ./kubectl /usr/local/bin/kubectl
```

minikube installation

```
curl -LO https://storage.googleapis.com/minikube/releases/v1.12.2/minikube_latest_amd64.deb

sudo dpkg -i minikube_latest_amd64.deb
```

Note: we need to install minikube version 1.12.2 for ingress support in driver=none.

kubernetes installation

The recent kubernetes versions are,

- v1.18.x
- v1.17.x
- v1.16.x

We are running/installing minikube in root user.

```
sudo -i

minikube start --kubernetes-version=v1.18.1 --driver=none
```

It may take 5-10 mins, depends on your internet speed.

```
minikube start --help
```

Verification

```
minikube status
```

Verify the basic kubernetes CLI commands

```
kubectl get nodes

kubectl get pods
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

kubeconfig & kubectl

The Kubernetes Cluster Access credentials are stored in kubeconfig file. A file that is used to configure access to clusters is called a kubeconfig file. This is a generic way of referring to configuration files.

The file contains Kubernetes API Server address, certificates.

Kubectl is a command line tool to control your kubernetes cluster.

kubectl looks for a file named config in the $HOME/.kube directory. You can specify other kubeconfig files by setting the KUBECONFIG environment variable .

As part of our minikube installing, minikube creates $HOME/.kube file.

```
export KUBECONFIG=dev-cluster.yaml

kubectl get pods
```

kubectl command overview

```
kubectl [command] [TYPE] [NAME] [flags]
```

command (operations) :

- get
- delete
- create
- edit
- apply
- describe
- ..... etc...

TYPE (Resource Types)

- pods
- deployments
- namespaces
- services
- ingress
- configmaps
- .....etc...

NAME means, Name of the resource

Flags : Specifies optional flag

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Example Commands:

```
kubectl get pods

kubectl get nodes

kubectl describe nodes ip-172-31-17-157
```

Reference:

- https://minikube.sigs.k8s.io/docs/
- https://kubernetes.io/docs/setup/release/version-skew-policy/
- https://github.com/kubernetes/kubernetes/releases
- https://kubernetes.io/docs/reference/kubectl/overview/
- https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

---

2. Basic Minikube commands

Operational Commands

Status

Gets the status of a local Kubernetes cluster.

```
minikube status
```

Pause/UnPause

pause and unpause only kubernetes control plane services. User plane(deployments) will continue to run, user can access.

```
minikube pause

minikube unpause
```

Stop/Start

Stop your local cluster & Start a cluster (Running)

```
minikube stop

minikube start
```

delete

```
minikube delete --all
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

#Upgrade to the specific version

```
minikube start --kubernetes-version=v1.18.2 --driver=none
```

Addons

minikube provides handy addon command, to install the necessary/basic important system deloyments required by the user .

Example:

- kubernetes dashboard
- ingress controller
- storage driver etc

list

```
minikube addons list
```

install

```
minikube addons enable dashboard
```

remove

```
minikube addons disable dashboard
```

Metrics Server

1. Install kubernetes metrics server

```
minikube addons enable metrics-server
```

2. Check the pods & services of dashboard

```
kubectl get pods -A
kubectl get svc -A
```

3. verify the metrics

```
kubectl top nodes
kubectl top pods
```

Dashboard Demo

1. Install the kubernetes dashboard (UI)

```
minikube addons enable dashboard
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

2. Check the pods & services of dashboard

```
kubectl get pods -A
kubectl get svc -A
```

3. port forwarding

```
kubectl port-forward --address 0.0.0.0 --namespace kubernetes-dashboard svc/kubernetes-dashboard 8080:80
```

Here 80 is a service port, 8080 is a forward port.

Open the browser and access it from your laptop. http://x.x.x.x:8080

That's all.

Reference

- https://minikube.sigs.k8s.io/docs/
- https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
- https://github.com/kubernetes/dashboard
- https://github.com/kubernetes-sigs/metrics-server

---

3. Kubernetes Resources

Sample Deployment

1. create nginx-web.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  labels:
    app: nginx-svc
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
---
```

2. Apply this yaml file

```
kubectl apply -f nginx-web.yaml
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

3. verify the resources

```
kubectl get deployments
kubectl get services
```

4. open the service nodeport assigned for this service in AWS Security Group

5. Now access the NGINX Webserver default page from your Laptop

```
http://aws-ip:3xxxx
```

Pod

1. create a nginx-pod.yaml file

```
kind: Pod
apiVersion: v1
metadata:
  name: nginxpod
  labels:
    foo: bar
spec:
  containers:
  - name: nginxpod
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

2. create a resource

```
kubectl apply -f nginx-pod.yaml
```

3. verify the operations

```
kubectl get pods
kubectl describe pods nginxpod
```

4. verify the logs

```
kubectl logs -f nginxpod
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

5. login to the nginx pod shell, and execute some commands.

```
kubectl exec -it nginxpod -- /bin/sh
ls -lrt
```

6. port forwarding to access from outside(not Recommended - only for debugging)

```
kubectl port-forward --address 0.0.0.0  pod/nginxpod 8080:80
```

7. Delete the pod

```
kubectl delete -f nginx-pod.yaml
#or
kubectl delete pod nginxpod
```

Deployment

1. Create an nginx deploy yaml(nginx-deploy.yaml) file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
    ports:
    - containerPort: 80
```

2. Create a resource

```
kubectl apply -f nginx-deploy.yaml
```

3. Verify the operations

```
kubectl get deployment
kubectl describe deployment nginx-deployment

kubectl get pods

kubectl describe pods nginxpod
```

4. Verify the pod logs
5. Login to the each nginx pod shell, and execute some commands.
6. Now delete some pod...

```
kubectl delete pod nginx-pod
```

Now the pods will be created automatically and keep the replication number as specified in the deployment.

7. Delete the deployment

```
kubectl delete deployment nginx-deployment
#or

kubectl delete -f nginx-deploy.yaml
```

Services

There are 3 types of services.

Reference: https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

Expose a Service onto an external IP address. Kubernetes supports two ways of doing this: **NodePorts and LoadBalancers**

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

NodePort

1. Deploy the nginix-deploy.yaml file (previous step)
2. Create a service for nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  labels:
    app: nginx-svc
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
```

targetPort: is the port the container accepts traffic on, **port**: is the abstracted Service port, which can be any port other pods use to access the Service

The default Nodeport ports range are 30000-32767

2. Apply this Resource

```
kubectl apply -f nginx-deploy.yaml
```

3. Verify it

```
kubectl get services
```

Note down the exposed port. Open that port from your security group of the VM.

Open a browser from your laptop and access it.

```
http://54.255.208.88:3xxxx/
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

4. Delete the Resources

```
kubectl delete -f  nginx-deploy.yaml
kubectl delete -f  nginx-svc.yaml
```

ClusterIP

ClusterIP: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.

1. Deploy the nginix-deploy.yaml file (previous step)
2. Create a service for nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  labels:
    app: nginx-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: nginx
```

3. Apply this Resources

```
kubectl apply -f nginx-svc.yaml
```

4. Create one more client deployment (client.yaml)

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-client
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sample-client
  template:
    metadata:
      labels:
        app: sample-client
    spec:
      containers:
      - image: sureshkvl/nginx-test:1.0
        name: sample-client
---


kubectl apply -f client.yaml
```

5. Now from client pod, request the nginx service

```
kubectl exec -it sample-client-xxxxxx -- /bin/sh
curl http://nginix-svc
```

This service is not exposed outside

LoadBalancer

On cloud providers which support external load balancers (such as amazon elb), setting the type field to LoadBalancer provisions a load balancer for your Service.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Ingress Controller deployment

1. Install ingress controller

```
minikube addons list
minikube addons enable ingress
```

2. verify the pods,svc,ingress

```
kubectl get pods -A
kubectl get svc -A
```

Now ingress controller is setup, we can consume through ingress resource.

Ingress Resource

Ingress manages external access to the services in a cluster (HTTP & HTTPS).

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

An Ingress does not expose arbitrary ports or protocols (only HTTP or HTTPS)

Prerequisites: Ingress Controller must be deployed.

References: https://kubernetes.io/docs/concepts/services-networking/ingress/

Example resource

Example1

1. Deploy the nginx-deploy.yaml
2. Deploy the nginx-service.yaml(ClusterIP)
3. Create the nginx-ingress.yaml file (ingress resource for nginx service)

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
spec:
  rules:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  - host: web.54.255.208.88.xip.io
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx-svc
          servicePort: http
---
```

4. Verify the resources.

```
kubectl get ingress
kubectl describe ingress nginx-ingress
```

5. Now access the URL from your laptop

```
http://web.54.255.208.88.xip.io
```

Namespace

Namespaces are a way to divide cluster resources between multiple users.

Namespaces are intended for use in environments with many users spread across multiple teams, or projects.

Kubernetes control plane components are deployed in **kube-system** namespace.

The default namespace name is **default**.

To display, all namespaces available in the cluster

```
kubectl get  namespaces
#or
kubectl get ns
```

To create a namespace

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
kubectl create namespace test
```

Create a deployment (nginx-deployment) in a test namespace

```
kubectl apply -f nginx-deployment.yaml  -n test
```

To display the resources (pods, deployments) in the namespaces, use "-n namespace-name"

```
kubectl get pods

kubectl get pods -n kube-system

kubectl get pods -n test

kubectl get deployments -n test
```

References

- https://en.wikipedia.org/wiki/YAML
- https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/
- https://kubernetes.io/docs/concepts/workloads/
- https://kubernetes.io/docs/concepts/services-networking/
- https://kubernetes.io/docs/concepts/storage/
- https://kubernetes.io/docs/concepts/configuration/
- https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html
- https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
- https://kubernetes.github.io/ingress-nginx/

---

4. DNS Setup

DNS

Option1:

1. If you have valid internet domain name (purchased from domain name vendor such as godaddy), point the domain DNS Record to this public of this VM.

Example: mydomain.com

2. We can create all deployments with this domain name.

   Example:

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

- wordpress.mydomain.com
- db.mydomain.com
- jenkins.mydomain.com
-

Option2:

1. If you don't have valid domain name, still we can use the public IP address (18.221.195.181) as domain name, using special testing method http://54.255.208.88.xip.io

2. We can create all deployments with this domain name. Example:

    - wordpress.54.255.208.88.xip.io
    - db.54.255.208.88.xip.io
    - jenkins.54.255.208.88.xip.io

---

5. Deployments with Manifests file

Example1 - Simple WebServer (blue)

Minimum Resources - deployment, service, ingress

1. Create the webblue.yaml file

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-blue
  labels:
    app: web-blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-blue
  template:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
    metadata:
      labels:
        app: web-blue
    spec:
      containers:
      - name: color
        image: knet/color:3
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-blue-svc
  labels:
    app: web-blue-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: web-blue
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  name: blue-ingress
spec:
  rules:
  - host: blue.knetsks.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web-blue-svc
          servicePort: http
---
```

Note: CHANGE your sub domain name **- host: blue.knetsks.com** according to your domain name or public IP (blue.54.255.208.88.xip.io)

Apply it.

```
kubectl apply -f webblue.yaml
```

2. Verify the deployments, service, ingress

```
kubectl get deploy
kubectl get pods
kubectl get ingress
```

3. Open the URL in your Laptop browser

```
http://blue.knetsks.com
```

Example2 - Simple WebServer (Red background)

Minimum Resources - deployment, service, ingress

```
---
apiVersion: apps/v1
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
kind: Deployment
metadata:
  name: web-red
  labels:
    app: web-red
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-red
  template:
    metadata:
      labels:
        app: web-red
    spec:
      containers:
      - name: color
        image: knet/color:1
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-red-svc
  labels:
    app: web-red-svc
spec:
  type: ClusterIP
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: web-red
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: red-ingress
spec:
  rules:
  - host: red.knetsks.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web-red-svc
          servicePort: http
---
```

Same as example1

Example3 - Simple WebServer (green background)

Minimum Resources - deployment, service, ingress

```
---
apiVersion: apps/v1
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
kind: Deployment
metadata:
  name: web-green
  labels:
    app: web-green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-green
  template:
    metadata:
      labels:
        app: web-green
    spec:
      containers:
      - name: color
        image: knet/color:2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-green-svc
  labels:
    app: web-green-svc
spec:
  type: ClusterIP
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: web-green
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: green-ingress
spec:
  rules:
  - host: green.knetsks.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web-green-svc
          servicePort: http
---
```

Same as example1

---

7. HELM

Installation

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

For helm 3.0, we do not required any server side installation in kubernetes.

1. Install helm client (3.x) & Update the helm repo

```
wget https://get.helm.sh/helm-v3.2.1-linux-amd64.tar.gz
tar xvzf helm-v3.2.1-linux-amd64.tar.gz

cp linux-amd64/helm /usr/bin/.

helm version
```

Helm version: https://github.com/helm/helm/releases

2. Repo Update

```
helm repo list
helm repo add stable https://kubernetes-charts.storage.googleapis.com/

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update
```

Helm Commands

Search Charts/Applications from the repo

```
helm repo list

helm search repo stable

helm search repo stable | grep mysql

helm search repo stable/my
```

Get details of specific chart

```
helm show chart stable/mysql

helm show readme stable/mysql

helm show values stable/mysql

helm show all stable/mysql
```

dry-run to see the resources for this chart.

```
helm install --dry-run  db1 stable/mysql
```

Install & Uninstall

```
helm install db1 stable/mysql
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
helm history db1

helm uninstall db1

helm list
```

Customize the chart values

- Every chart has comes with default config values which can be customized by the user.
- User can override this default values by providing the explicit values, during installation.

Identify the customization values

- Refer the respective github page of the chart, and read the **values.yaml** file. Example : https://github.com/bitnami/charts/tree/master/bitnami/wordpress
- Run the below show values command to see the values.

```
helm show values bitnami/wordpress
```

- Create a customization values in yaml format. pass this file as input during installation

---

Wordpress Deployment

1. Create a wordpress namespace

```
kubectl create ns wordpress
```

2. Read the configuration parameter for wordpress from bitname/wordpress chart.

3. Create a values.yaml file

- Install the Chart with this values.yaml file

```
wordpressUsername: admin
wordpressPassword: admin123

service:

  type: ClusterIP

persistence:

  enabled: false

mariadb:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  master:
    persistence:
      enabled: false
ingress:
  enabled: true
  hostname: wordpress.knetsks.com
```

4. Install it

```
helm install -n wordpress -f values.yaml wp1 bitnami/wordpress
```

5. Verify the deployments

```
kubectl get pods -n wordpress
kubectl get svc -n wordpress
kubectl get ingress -n wordpress
```

6. Open the browser and access the below url

```
http://wordpress.knetsks.com
```

7. Delete

```
helm delete -n wordpress wp1
```

Jenkins deployment

1. Create a cicd namespace

```
kubectl create ns cicd
```

2. Read the configuration parameter for wordpress from bitnami/jenkins chart.

3. Create a values.yaml file

- Install the Chart with this values.yaml file

```
service:
  type: ClusterIP
persistence:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  enabled: false
ingress:
  enabled: true
  hostname: jenkins.knetsks.com
```

4. Install it

```
helm install -n cicd -f values.yaml jenkins1 bitnami/jenkins
```

5. The output shows username & password for this jenkins deployment.

6. Verify the deployments

```
kubectl get pods -n cicd
kubectl get svc -n cicd
kubectl get ingress -n cicd
```

7. Open the browser and access the below url

```
http://jenkins.knetsks.com
```

8. Use username/password to login

Redmine deployment

1. Read the configuration parameter for wordpress from bitnami/redmine.

2. Create a values.yaml file

- install the Chart with this values.yaml file

```
replicas: 2
service:
  type: ClusterIP
persistence:
  enabled: false
mariadb:
  master:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  persistence:
    enabled: false
ingress:
 enabled: true
 hostname: redmine.knetsks.com
```

3. Install it

```
helm install -f values.yaml redmine bitnami/redmine
```

---

8. Monitoring

Deploy the Monitoring Stack

1. Create a monitoring namespace

```
kubectl create ns monitoring
```

2. Create values.yaml file

```
prometheus:
 prometheusSpec:
   retention: 1h
 ingress:
   enabled: true
   hosts: [prometheus.knetsks.com]
alertmanager:
 ingress:
   enabled: true
   hosts:  [alerts.knetsks.com]
grafana:
 ingress:
   enabled: true
   hosts: [grafana.knetsks.com]
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

To know more details about configuration parameters, please refer
https://github.com/helm/charts/tree/master/stable/prometheus-operator

3. Install it

```
helm install -n monitoring -f values.yaml mon stable/prometheus-operator
```

4. Verify the Pods, Services, Ingress

```
kubectl get pods -n monitoring
kubectl get svc -n monitoring

kubectl get ingress -n monitoring
```

Wait, till Pods are Active

5. Prometheus portal http://prometheus.knetsks.com

6. AlertManager portal http://alerts.knetsks.com

7. Grafana portal http://grafana.knetsks.com

   Default username: admin
   Password: prom-operator

Verify the Cluster Metrics in Grafana Dashboard

The default prometheus operator installation have Kubernetes cluster metrics dashboard (infrastructure).

Todo

Monitor your Application (POD) Metrics

In the earlier chapter, we have seen the metrics of kubernetes cluster components (Infrastructure).

Now, we would like to monitor the user deployment applications such as wordpress, or other components.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Steps

1. Metrics exporter (AKA node exporter) for the application to be deployed as side car container. Usually this is supported/provided by the HELM Charts for all generic applications. If not available, we have to make it.

2. Create the Prometheus Service Monitor Resource (This resource links/routes the metrics to our Prometheus Monitoring Solution)

3. Verify the Metrics are visible in Prometheus & Grafana UI.

4. Make a Dashboard for this metrics in Grafana.

Example1: Wordpress deployment

I have deleted our existing wordpress deployment. and creating new one with metrics enabled.

1. Create values.yaml file

```
wordpressUsername: admin
wordpressPassword: admin123
service:
  type: ClusterIP
persistence:
  enabled: false
mariadb:
  master:
    persistence:
      enabled: false
ingress:
  enabled: true
  hostname: wordpress.knetsks.com
metrics:
  enabled: true
```

2. Install it

```
helm install -n wordpress -f values.yaml wp1 bitnami/wordpress
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

3. Verify the metrics are exposed from the pods

```
helm list -n wordpress

kubectl get pods -n wordpress

kubectl describe pod/wp1-wordpress-fc86478f9-jx9kg -n wordpress
```

Verify the side-car (apache exporter) container also running in the wordpress pods, this expose /metrics on port 9117/TCP

Do, kubectl port forwarding and verify it.

```
kubectl port-forward -n wordpress  pod/wp1-wordpress-fc86478f9-jx9kg 9117:9117
```

From the VM,

```
curl localhost:9117/metrics

.....

# TYPE apache_accesses_total counter

apache_accesses_total 289

..... skipped
```

4. Verify the metrics are exposed from the service

```
kubectl get svc -n wordpress
```

We should see the service expose the 9117/TCP Port also. Now we should port forward the service and check this.

```
kubectl port-forward -n wordpress  svc/wp1-wordpress 9117:9117
```

Do query the metrics endpoint

```
curl localhost:9117/metrics

.....

# TYPE apache_accesses_total counter

apache_accesses_total 289

..... skipped
```

So metrics are exposed from service also..

5. Now, create the service monitor

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Service monitor resource requires 2 important parameters

a) ServiceMonitorSelector, ServiceMonitorNameSpaceSelector parameters to be noted from Prometheus resource.

```
kubectl -n monitoring get prometheus -o yaml
Example:
....
....skipped
  serviceMonitorNamespaceSelector: {}
  serviceMonitorSelector:
      matchLabels:
      release: mon;
```

Here "release:mon" is ServiceMonitorSelector Lablel, serviceMonitorNamespaceSelector is {}, which means default namespace. This means, Prometheus can query the metrics only from default name space.

b) WordPress service label name, and metrics port name to be noted from WordPress service resource

```
kubectl describe svc/wp1-wordpress -n wordpress
```

c) Create service monitor (sm.yaml) yaml file.

```
---
kind: ServiceMonitor
apiVersion: monitoring.coreos.com/v1
metadata:
  name: wordpress-sm
  labels:
    app: wordpress-sm
    release: mon
spec:
  selector:
    matchLabels:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
    app.kubernetes.io/instance: wp1
    app.kubernetes.io/name: wordpress
  namespaceSelector:
    matchNames:
    - wordpress
  endpoints:
  - port: metrics
    interval: 10s
---
```

Note:

- lables - "release:mon", we have refereed from Prometheus resource
- matchLabels - we have referred from the kubernetes service resource
- matchNames - WordPress. We run our service in WordPress namespace.

```
kubectl apply -f sm.yaml -n wordpress
```

6. Now verify the Service Monitor

```
kubectl get servicemonitor -n wordpress

kubectl describe servicemonitor wordpress-sm  -n wordpress
```

7. Verify it in the Prometheus UI

- Open Prometheus UI
- Type any parameter which you seen in /metrics endpoint ex: apache_accesses_total
- Click "Execute", you will see the metric.

8. Verify in Grafana

- Open Grafana UI , and click EXPLORE
- Type any parameter which you seen in /metrics endpoint ex: apache_accesses_total
- Click "Run Query", you will see the graph.
- search the apache dashboard from Grafana https://grafana.com/grafana/dashboards
- the dashboard link is, https://grafana.com/grafana/dashboards/3894
- Click + and Import
- Paste the link of dashboard and click Load

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Logging

Centralized log management and analysing is important requirements for Containerized environment.

Logs from Kubernetes control plane components (api,scheduler,controller, etc) and user application deployments(such as WordPress, Jenkins, other applications.) to be collected and visualized in centralized User Interface (dashboard).

LOKI is log aggregation system inspired by Prometheus. So it works out of the box.

It requires 3 components

- loki
- promtail
- grafana

We already deployed Grafana as part of monitoring stack.

loki-stack helm chart deploys loki and promtail.

1. Repo update

```
helm repo add loki https://grafana.github.io/loki/charts

helm repo update
```

2. Install loki-stacl

loki-stack is included with loki and promtail.

```
kubectl create ns monitoring

helm install  loki  loki/loki-stack -n monitoring
```

3. How to verify

```
k get pods -n monitoring

k get svc -n monitoring
```

4. Integrate Loki with Grafana

- open Grafana UI (grafana.knetsks.com)
- click Configuration -> Datasources
- Add Datasource
- Select loki
- In the name specify "loki",and in URL "http://loki.monitoring:3100". Note this is service name of the loki "k get svc -n monitoring"

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

- Save and Test. It should work.

5. How to view the Logs.

- Open Grafana UI (grafana.knetsks.com)
- Click Explore , and select Loki Datasource
- Click LogLabels, you can see the query filter based on "namespace", "pod", "job", "container" etc....

References

- https://github.com/grafana/loki/tree/master/production/helm/loki
- https://github.com/grafana/loki
- https://www.scaleway.com/en/docs/use-loki-to-manage-k8s-application-logs/

---

9. Certificate Manager Integration

So far, we used HTTP for our services, such as (http://grafana.knetsks.com, http://wordpress.knetsks.com) etc. Because, we don't have valid certificate management system for our services.

We can deploy **cert-manager** and **lets-encrypt**, for certificate management system. This components will take care of HTTPS (Certificates) for our services automatically.

Prerequisites:

- ingress controller (we have already deployed ingress-controller)

Deploy cert-manager

1. Create a namespace

```
kubectl create ns cert-manager
```

2. Install custom resources

```
kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/releases/download/v0.14.1/cert-manager.crds.yaml
```

3. Update the helm repo for certmanager

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

4. Create values.yaml

```
ingressShim:
  defaultIssuerName: letsencrypt-prod

  defaultIssuerKind: ClusterIssuer
```

Note: **letsencrypt-staging** or **letsencrypt-prod** can be used as defaultIssuerName. If it is prod, certificate will be valid (no security risk in browser)

5. Install

```
helm install cert-manager jetstack/cert-manager --version v0.14.1 --namespace cert-manager   -f values.yaml
```

6. Verification

```
kubectl get pods --namespace cert-manager
```

Cert manager can be configured with variety of certificate issuers, such as

- SelfSigned
- CA
- Vault
- Venafi
- External
- ACME

We are going to see Automated Certificate Management Environment (ACME) method.

Lets-encrypt is Free, Opencertificate Authority (ACME). We are going to use lets-encrypt.

Integrate cert-manager with Lets-encrypt certissuer

There are two standard methods for validation (RFC 8555) of Domain name by ACME (To make sure, client is requesting the certificate for his own domain)

- http1
- dns1

HTTP01

1. Create a cluster issuer (cissuer.yaml) yaml

```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: knetsolutions2@gmail.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

Note: Here your ingress class name is nginx. (This is default class name)

2. Apply and verify it

```
k apply -f cissuer.yaml -n cert-manager
 k get clusterissuer  -n cert-manager
```

That's all. It's ready, how we can consume.

3. Let's do the sample color deployment(color.yaml).

In the ingress resource, add the annotations as below,

```
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
    cert-manager.io/acme-challenge-type: http01
```

The complete manifests file is below,

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
  name: web-blue
  labels:
    app: web-blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-blue
  template:
    metadata:
      labels:
        app: web-blue
    spec:
      containers:
      - name: color
        image: knet/color:3
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-blue-svc
  labels:
    app: web-blue-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: web-blue
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: blue-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
    cert-manager.io/acme-challenge-type: http01
spec:
  tls:
  - hosts: ["blue.knetsks.com"]
    secretName: blue-secret-name
  rules:
  - host: blue.knetsks.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web-blue-svc
          servicePort: http


kubectl apply -f color.yaml
```

That's all.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

4. WordPress deployment

```
wordpressUsername: admin
wordpressPassword: admin123
service:
  type: ClusterIP
persistence:
  enabled: false
mariadb:
  master:
    persistence:
      enabled: false
ingress:
  enabled: true
  hostname: wordpress.knetsks.com
  certManager: true
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    cert-manager.io/acme-challenge-type: http01
  tls: true
```

DNS01

There are only few DNS providers are supported by cert-manager (Route53, clourflare, akami, etc) https://cert-manager.io/docs/configuration/acme/dns01/

We demonstrate with AWS Route53 DNS.

Note: we use AWS access-key/secret key for domain name validation.

Prerequisites: You should specify the aws_access, aws_secret_key, hostedZoneID (for domain name)

1. Create secret (AWS secret jet)

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
kubectl create secret generic route53-secret  --from-literal=secret-access-key=xxx -n cert-manager
```

2. Create staging.yaml

```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: knetsolutions2@gmail.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - selector:
        dnsZones:
          - "knetsks.com"
      dns01:
        route53:
          region: eu-east-2
          hostedZoneID: xxxxxxxx
          accessKeyID: xxxxxxx
          secretAccessKeySecretRef:
            name: route53-secret
            key: secret-access-key
```

3. Apply staging.yml

```
kubectl apply -f staging.yml -n cert-manager
```

That's all its ready.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

How to consume?

1. In the ingress resource, use this annotations

```
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
```

2. Example with WordPress helm chart values

```
wordpressUsername: admin
wordpressPassword: admin123
service:
  type: ClusterIP
persistence:
  enabled: false
mariadb:
  master:
    persistence:
      enabled: false
ingress:
  enabled: true
  hostname: wordpress.knetsks.com
  certManager: true
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
  tls: true
```

3. Verification

```
kubectl get certificate -A
kubectl get Issuers,ClusterIssuers -A
kubectl get Certificates,CertificateRequests,Orders,Challenges -A
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

```
kubectl describe clusterissuer.cert-manager.io/letsencrypt-staging -A
```

References

- https://cert-manager.io/docs/
- https://letsencrypt.org/

---

10. Other topics

Installing Minikube in Laptop

The preferred option is install it in the VM.

1. Create a Ubuntu 18.04 VM
2. Login to the VM,
3. Install minikube with driver None.

As mentioned in section 2.

Other options are, 1. From your linux laptop, 2. Install minikube with driver=kvm2

```
minikube start --kubernetes-version=v1.18.1 --driver=kvm2
```

Default username/password for the minikube vm is docker & tcuser

Command to delete the exited docker containers

```
sudo docker ps -a | grep Exit | cut -d ' ' -f 1 | xargs sudo docker rm
```

11. References

Docker Installation

- https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04

Minikube

- https://minikube.sigs.k8s.io/docs/

Dashboard

- https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Kubernetes

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

- https://kubernetes.io/docs/tasks/tools/install-kubectl/

Jenkins chart

- https://github.com/bitnami/charts/tree/master/bitnami/jenkins

Prometheus-operator

- https://github.com/helm/charts/tree/master/stable/prometheus-operator

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Practical_Kubernetes_by_Stanley.pdf
Email: s.stanley.mca@gmail.com