**FREE eBook**

# LEARNING kubernetes

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#kubernetes

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: kubernetes

It is an unofficial and free kubernetes ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official kubernetes.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with kubernetes

## Remarks

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

With Kubernetes, you are able to quickly and efficiently respond to customer demand:

- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Seamlessly roll out new features.
- Optimize use of your hardware by using only the resources you need.

**Why do I need Kubernetes and what can it do?**

Kubernetes can schedule and run application containers on clusters of physical or virtual machines. However, Kubernetes also allows developers to 'cut the cord' to physical and virtual machines, moving from a **host-centric** infrastructure to a **container-centric** infrastructure, which provides the full advantages and benefits inherent to containers. Kubernetes provides the infrastructure to build a truly container-centric development environment.

Kubernetes satisfies a number of common needs of applications running in production, such as:

- co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model,
- mounting storage systems,
- distributing secrets,
- application health checking,
- replicating application instances,
- horizontal auto-scaling,
- naming and discovery,
- load balancing,
- rolling updates,
- resource monitoring,
- log access and ingestion,
- support for introspection and debugging, and
- identity and authorization.

This provides the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and facilitates portability across infrastructure providers.

## Versions

| Version | Release Date |
| --- | --- |
| 1.7 | 2017-06-28 |
| 1.6 | 2017-02-22 |
| 1.5 | 2016-12-13 |
| 1.4 | 2016-09-26 |
| 1.3 | 2016-07-06 |
| 1.2 | 2016-03-17 |
| 1.1 | 2015-09-09 |
| 1.0 | 2015-07-18 |

## Examples

### Installing Minikube

Minikube creates a local cluster of virtual machines to run Kubernetes on.It is the simplest method to get your hands dirty with Kubernetes on your local machine.

Documentation for Minikube can be found at http://kubernetes.io/docs/getting-started-guides/minikube/

# Requirements

- On macOS, xhyve driver, VirtualBox or VMware Fusion hypervisors
- On Linux, VirtualBox or KVM hypervisors
- On Windows VirtualBox or Hyper-V hypervisors
- VT-x/AMD-v virtualization enabled

To check if virtualization support is enabled, run the appropriate command from below. The command will output something if virtualization is enabled.

```
# On Linux
cat /proc/cpuinfo | grep 'vmx\|svm'
# On OSX
sysctl -a | grep machdep.cpu.features | grep VMX
```

# Installation

Minikube is a single binary. Thus, installation is as easy as downloading the binary and placing it

in your path.

```
# Specify the version of minikube to download.
# Latest version can be retrieved from
# https://github.com/kubernetes/minikube/releases
VERSION=v0.16.0

# If on Linux
OS=linux
# If on OSX
# OS=darwin

# URL to download minikube binary from
URL=https://storage.googleapis.com/minikube/releases/$VERSION/minikube-$OS-amd64

# Download binary and place in path.
curl -Lo minikube $URL
chmod +x minikube
sudo mv minikube /usr/local/bin/
```

# Usage

To start a new cluster:

```
minikube start
```

This will create a new cluster of local virtual machines with Kubernetes already installed and configured.

You can access the Kubernetes dashboard with:

```
minikube dashboard
```

Minikube creates a related context for `kubectl` which can be used with:

```
kubectl config use-context minikube
```

Once ready the local Kubernetes can be used:

```
kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
kubectl expose deployment hello-minikube --type=NodePort
curl $(minikube service hello-minikube --url)
```

To stop the local cluster:

```
minikube stop
```

To delete the local cluster, note new IP will be allocated after creation:

```
minikube delete
```

### Install on Google Cloud

Kubernetes was originally developed by Google to power their Container Engine. As such, Kubernetes clusters are a first class citizen at Google.

Creating a Kubernetes cluster in the container engine requires `gcloud` command from the Google Cloud SDK. To install this command locally, use one of the following options:

- use the interactive installer (the easiest way for the newcomers):

```
curl https://sdk.cloud.google.com | bash
exec -l $SHELL
gcloud init
```

- download the SDK from https://cloud.google.com/sdk/ and run the appropriate install file.

  For example, to install in Linux (x86_64):

```
curl -Lo gcloud-sdk.tar.gz https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-142.0.0-linux-x86_64.tar.gz
tar xvf ./gcloud-sdk.tar.gz
./google-cloud-sdk/install.sh
gcloud init
```

Once `gcloud` is installed, create a Kubernetes cluster with:

```
# Give our cluster a name
CLUSTER_NAME=example-cluster

# Number of machines in the cluster.
NUM_NODES=3

gcloud container clusters create $CLUSTER_NAME --num_nodes=$NUM_VMS
```

### Configure kubectl

A Kubernetes cluster is controlled using the `kubectl` command. The method of configuring `kubectl` depends on where Kubernetes is installed.

## Google Cloud (Container Engine)

To install kubectl using the Google Cloud SDK:

```
gcloud components install kubectl
```

To configure kubectl to control an existing Kubernetes cluster in Container Engine:

```
gcloud container clusters get-credentials $CLUSTER_NAME
```

# Minikube

When using minikube, the kubectl binary needs to be manually downloaded and placed in the path.

```
# Version of Kubernetes.
K8S_VERSION=$(curl -sS https://storage.googleapis.com/kubernetes-release/release/stable.txt)
# Operating System. Can be one of {linux, darwin}
GOOS=linux
# Architecture. Can be one of {386, amd64, arm64, ppc641e}
GOARCH=amd64

# Download and place in path.
curl -Lo kubectl http://storage.googleapis.com/kubernetes-
release/release/${K8S_VERSION}/bin/${GOOS}/${GOARCH}/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

The minikube binary automatically configures kubectl when starting a cluster.

```
minikube start
# kubectl is now ready to use!
```

## Kubectl in command line

After you have a running cluster you can manage it with the `kubectl` command. Most of the commands you can get with the `kubectl --help` command, but I show you the most common commands, for manage and getting info about your cluster, nodes, pods, services and labels.

---

For getting information about the cluster you can user the following command

```
kubectl cluster-info
```

It will show you the running address and port.

---

For getting short information about the nodes, pods, services, etc. or any resources which got a place on the cluster you can use the following command

```
kubectl get {nodes, pods, services, ...}
```

The output mostly one line per resource.

---

For getting detailed description about the resources you can use the `describe` flag for the `kubectl`

```
kubectl describe {nodes, pods, ...}
```

---

The deployed apps are only visible inside the cluster, so if you want to get the output from outside the cluster you should create a route between the terminal and kubernetes cluster.

```
kubectl proxy
```

It will open a API, where we can get everything from the cluster. If you want to get the name of the pods for getting information about, you should use the following command:

```
kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}'
```

It will list the pods for later usage.

```
curl http://localhost:8001/api/v1/proxy/namespaces/default/pods/{pod_name}/
```

---

Two other common command is the getting logs and the execute a command from/in the containerized app.

```
kubectl logs {pod_name}
kubectl exec {pod_name} {command}
```

---

Configuring tab completion for your shell can be done with:

```
source <(kubectl completion zsh)   # if you're using zsh
source <(kubectl completion bash)  # if you're using bash
```

or more programatically:

```
source <(kubectl completion "${0/-/}")
```

## Hello World

Once your Kubernetes cluster is running and `kubectl` is configured you could run your first application with a few steps. This can be done using the imperative commands which doesn't need configuration files.

In order to run an application you need to provide a deployment name (`bootcamp`), the container image location (`docker.io/jocatalin/kubernetes-bootcamp:v1`) and the port (`8080`)

```
$ kubectl run bootcamp --image=docker.io/jocatalin/kubernetes-bootcamp:v1 --port=8080
```

Confirm that it worked with:

```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bootcamp               1         1         1            1          6s
```

To expose your application and make it accessible from the outside run:

```
$ kubectl expose deployment/bootcamp --type="LoadBalancer" --port 8080
```

Confirm that it worked with:

```
$ kubectl get services
NAME                  CLUSTER-IP   EXTERNAL-IP         PORT(S)    AGE
kubernetes            10.0.0.1     <none>              443/TCP    3m
bootcamp              10.3.245.61  104.155.111.170     8080:32452/TCP   2m
```

To access the services, use the external IP and the application port e.g. like this:

```
$ export EXTERNAL_IP=$(kubectl get service bootcamp --
output=jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ export PORT=$(kubectl get services --output=jsonpath='{.items[0].spec.ports[0].port}')
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-2fhnk | v=1
```

The same could be done manually with the data provided in:

```
$ kubectl describe service bootcamp
Name:                   bootcamp
Namespace:              default
Labels:                 run=bootcamp
Selector:               run=bootcamp
Type:                   LoadBalancer
IP:                     10.3.245.61
LoadBalancer Ingress:   104.155.111.170
Port:                   <unset> 8080/TCP
NodePort:               <unset> 32452/TCP
Endpoints:              10.0.0.3:8080
... events and details left out ....

$ export NODE=104.155.111.170
$ export PORT=8080
```

Once this worked you can scale up your application with:

```
$ kubectl scale deployments/bootcamp --replicas=4
```

And check the result with:

```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bootcamp               4         4         4            4         30s

$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-2fhnk | v=1
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-gmtv5 | v=1
```

Mind the changing pod id.

In order to push out a new application version run:

```
kubectl set image deployments/bootcamp bootcamp=jocatalin/kubernetes-bootcamp:v2
```

And confirm it with:

```
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-284539476-gafwev3 | v=2
```

Cleaning up is finally done with:

```
$ kubectl delete deployment bootcamp
$ kubectl delete service bootcamp
```

Read Getting started with kubernetes online: https://riptutorial.com/kubernetes/topic/4099/getting-started-with-kubernetes

# Chapter 2: Calling Kubernetes API

## Examples

### Using Kubernetes Go Client - Outside of Cluster

```
package main

import (
  "fmt"

  "k8s.io/client-go/1.5/kubernetes"
  "k8s.io/client-go/1.5/pkg/api/v1"
  "k8s.io/client-go/1.5/tools/clientcmd"
)

func main()  {
    config, err := clientcmd.BuildConfigFromFlags("", <kube-config-path>)
    if err != nil {
      return nil, err
    }

    c, err := kubernetes.NewForConfig(config)
    if err != nil {
      return nil, err
    }

    // Get Pod by name
    pod, err := c.Pods(v1.NamespaceDefault).Get("my-pod")
    if err != nil {
        fmt.Println(err)
        return
    }

    // Print its creation time
    fmt.Println(pod.GetCreationTimestamp())
}
```

### List replicasets by given deployment with kubernetes go client

```
package main

import (
    "k8s.io/kubernetes/pkg/api"
    unver "k8s.io/kubernetes/pkg/api/unversioned"
    "k8s.io/kubernetes/pkg/apis/extensions"
    "k8s.io/kubernetes/pkg/client/restclient"
    client "k8s.io/kubernetes/pkg/client/unversioned"
    "log"
    "os"
    "strings"
)

var logger *log.Logger
```

```
const (
    SERVER             string = "http://172.21.1.11:8080"
    RevisionAnnotation        = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func getReplicaSetsByDeployment(c *client.Client, deployment *extensions.Deployment)
([]extensions.ReplicaSet, error) {

    namespace := deployment.Namespace
    selector, err := unver.LabelSelectorAsSelector(deployment.Spec.Selector)
    if err != nil {
        return nil, err
    }
    options := api.ListOptions{LabelSelector: selector}
    rsList, err := c.Extensions().ReplicaSets(namespace).List(options)

    return rsList.Items, nil
}

func getDeploymentByReplicaSet(namespace string, c *client.Client, rs *extensions.ReplicaSet)
([]extensions.Deployment, error) {

    selector, err := unver.LabelSelectorAsSelector(rs.Spec.Selector)
    if err != nil {
        return nil, err
    }
    options := api.ListOptions{LabelSelector: selector}
    dps, err := c.Extensions().Deployments(namespace).List(options)
    if err != nil {
        return nil, err
    }

    return dps.Items, nil
}

func getDeploymentByReplicaSetName(namespace string, c *client.Client, rs
*extensions.ReplicaSet) (*extensions.Deployment, error) {

    name := rs.Name
    index := strings.LastIndex(name, "-")
    deploymentName := name[:index]

    dp, err := c.Extensions().Deployments(namespace).Get(deploymentName)
    if err != nil {
        return nil, err
    }

    return dp, nil
}

func main() {

    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
```

```
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    list, err := c.Extensions().Deployments(api.NamespaceDefault).List(api.ListOptions{})
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }

    logger.Printf("Deployment -------> ReplicaSet: ")

    for _, deployment := range list.Items {
        rses, err := getReplicaSetsByDeployment(c, &deployment)
        if err != nil {
            logger.Fatalf("GetReplicaSetsByDeployment Error: err=%s\n", err)
        }

        for _, rs := range rses {
            logger.Printf("ReplicaSet assioated with Deployment: rs-name=%s, revision=%s, dp-
name=%s\n",
                rs.Name, rs.Annotations[RevisionAnnotation], deployment.Name)
        }
    }

    logger.Printf("\n\nReplicaSet -------> Deployment: ")
    rsList, err := c.Extensions().ReplicaSets(api.NamespaceDefault).List(api.ListOptions{})
    if err != nil {
        log.Fatalf("Could not list ReplicaSet")
    }

    for _, rs := range rsList.Items {
        dp, err := getDeploymentByReplicaSetName(api.NamespaceDefault, c, &rs)
        if err != nil {
            logger.Fatalf("GetDeploymentByReplicaSet Error: err=%s\n", err)
        }

        logger.Printf("Deployment assioated with ReplicaSet: rs-name=%s, revision=%s, dp-
name=%s\n",
            rs.Name, rs.Annotations[RevisionAnnotation], dp.Name)
    }
}
```

## Rollback with the revision of replicasets using kubernetes go client

```
package main

import (
    //"k8s.io/kubernetes/pkg/api"
    "k8s.io/kubernetes/pkg/client/restclient"
    "log"
    "os"
    // unver "k8s.io/kubernetes/pkg/api/unversioned"
    "k8s.io/kubernetes/pkg/apis/extensions"
    client "k8s.io/kubernetes/pkg/client/unversioned"
)

var logger *log.Logger

var annotations = map[string]string{
```

```
    "Image":                      "nginx:1.7.9",
    "UserId":                     "2",
    "kubernetes.io/change-cause": "version mismatch",
}

const (
    DEPLOYMENT         string = "nginx-test"
    REVERSION          int64  = 4
    SERVER             string = "http://172.21.1.11:8080"
    RevisionAnnotation        = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func rollBack(c *client.Client, dp *extensions.Deployment, revision int64) error {

    dr := new(extensions.DeploymentRollback)
    dr.Name = dp.Name
    dr.UpdatedAnnotations = annotations
    dr.RollbackTo = extensions.RollbackConfig{Revision: revision}

    // Rollback
    err := c.Extensions().Deployments("ops").Rollback(dr)
    if err != nil {
        logger.Printf("Deployment Rollback Error: err=%s\n", err)
        return err
    }

    return nil
}

func main() {
    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    dp, err := c.Extensions().Deployments("ops").Get(DEPLOYMENT)
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }

    rollBack(c, dp, REVERSION)

}
```

## Rolling update with repliasets using kubernetes go client

```
package main

import (
    //    "k8s.io/kubernetes/pkg/api"
    // unver "k8s.io/kubernetes/pkg/api/unversioned"
```

```
    "k8s.io/kubernetes/pkg/apis/extensions"
    "k8s.io/kubernetes/pkg/client/restclient"
    client "k8s.io/kubernetes/pkg/client/unversioned"
    "k8s.io/kubernetes/pkg/util/intstr"
    "log"
    "os"
)

var logger *log.Logger

const (
    //DEPLOYMENT           string = "nginx-deployment"
    DEPLOYMENT           string = "nginx-test"
    RevisionHistoryLimit int32  = 5
    SERVER               string = "http://172.21.1.11:8080"
    RevisionAnnotation   string = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func rollingUpdate(c *client.Client, dp *extensions.Deployment) error {

    // New a DeploymentStrategy
    ds := new(extensions.DeploymentStrategy)
    ds.Type = extensions.RollingUpdateDeploymentStrategyType
    ds.RollingUpdate = new(extensions.RollingUpdateDeployment)
    ds.RollingUpdate.MaxUnavailable = intstr.FromInt(int(dp.Spec.Replicas))

    dp.Spec.Strategy = *ds

    // Image
    //dp.Spec.Template.Spec.Containers[0].Image = "nginx:1.9.7"
    dp.Spec.Template.Spec.Containers[0].Image = "nginx:1.9"

    // Update
    //_, err := c.Extensions().Deployments(api.NamespaceDefault).Update(dp)
    _, err := c.Extensions().Deployments("ops").Update(dp)
    if err != nil {
        logger.Printf("Update Deployment Error: err=%s\n", err)
        return err
    }
    return nil
}

func main() {
    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    //dp, err := c.Extensions().Deployments(api.NamespaceDefault).Get(DEPLOYMENT)
    dp, err := c.Extensions().Deployments("ops").Get(DEPLOYMENT)
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }
```

```
    rollingUpdate(c, dp)

}
```

## Using Kubernetes Go Client - Inside of Cluster

```
package main

import (
  "fmt"

  "k8s.io/client-go/1.5/kubernetes"
  "k8s.io/client-go/1.5/pkg/api/v1"
  "k8s.io/client-go/1.5/rest"
)

func main()  {
    config, err = rest.InClusterConfig()
    if err != nil {
      return nil, err
    }

    c, err := kubernetes.NewForConfig(config)
    if err != nil {
      return nil, err
    }

    // Get Pod by name
    pod, err := c.Pods(v1.NamespaceDefault).Get("my-pod")
    if err != nil {
        fmt.Println(err)
        return
    }

    // Print its creation time
    fmt.Println(pod.GetCreationTimestamp())
}
```

Read Calling Kubernetes API online: https://riptutorial.com/kubernetes/topic/5926/calling-kubernetes-api

# Chapter 3: Kubernetes in production

## Introduction

Introduce how to use kubernetes in production environment

## Examples

### Deploy zookeeper cluster in production using kubernetes and ceph

Dockerize zookeeper-3.4.6

---

Create a Dockerfile:

```
#########################################################
# Image: img.reg.3g:15000/zookeeper:3.4.6
#########################################################

FROM img.reg.3g:15000/jdk:1.7.0_67

MAINTAINER lth9739@gmail.com

USER root

ENV ZOOKEEPER_VERSION 3.4.6

ADD Dockerfile /

ADD zookeeper/ /opt/

COPY zoo.cfg /opt/zookeeper/conf/zoo.cfg

RUN mkdir -p /opt/zookeeper/{data,log}

WORKDIR /opt/zookeeper

VOLUME ["/opt/zookeeper/conf", "/opt/zookeeper/data", "/opt/zookeeper/log"]

COPY config-and-run.sh /opt/zookeeper/bin/

EXPOSE 2181 2888 3888

CMD ["/opt/zookeeper/bin/config-and-run.sh"]
```

See more details

Deploy zookeeper replica controller into kubernetes cluster

---

You can use this command to deploy the replica-controller of zookeeper:

```
kubectl create -f zookeeper-rc-1.json
```

```
{
  "apiVersion": "v1",
  "kind": "ReplicationController",
  "metadata": {
    "labels": {
      "component": "zookeeper"
    },
    "name": "zookeeper-1"
  },
  "spec": {
    "replicas": 1,
    "selector": {
      "server-id": "1",
      "role": "zookeeper-1"
    },
    "template": {
      "metadata": {
        "labels": {
          "server-id": "1",
          "role": "zookeeper-1"
        },
        "name": "zookeeper-1"
      },
      "spec": {
        "containers": [
          {
            "env": [
              {
                "value": "1",
                "name": "SERVER_ID"
              },
              {
                "value": "5",
                "name": "MAX_SERVERS"
              }
            ],
            "image": "img.reg.3g:15000/fabric8/zookeeper:latest",
            "name": "zookeeper-1",
            "ports": [
              {
                "containerPort": 2181,
                "name": "client",
                "protocol": "TCP"
              },
              {
                "containerPort": 2888,
                "name": "followers",
                "protocol": "TCP"
              },
              {
                "containerPort": 3888,
                "name": "election",
                "protocol": "TCP"
              }
            ],
            "volumeMounts": [
              {
                "mountPath": "/opt/zookeeper/data",
                "name": "zookeeper-1"
              }
            ]
```

```
          }
        ],
        "restartPolicy": "Always",
        "volumes": [
          {
            "name": "zookeeper-1",
            "rbd": {
              "monitors": [
                "10.151.32.27:6789",
                "10.151.32.29:6789",
                "10.151.32.32:6789"
              ],
              "pool": "rbd",
              "image": "log-zookeeper-1",
              "user": "admin",
              "secretRef": {
                "name": "ceph-secret-default"
              },
              "fsType": "ext4",
              "readOnly": false
            }
          }
        ]
      }
    }
  }
}
```

Deploy zookeeper service into kubernetes cluster

---

You can use this command to deploy the service of zookeeper:

```
kubectl create -f zookeeper-svc-1.json
```

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "zookeeper-1",
    "labels": {
      "name": "zookeeper-1"
    }
  },
  "spec": {
    "ports": [
      {
        "name": "client",
        "port": 2181,
        "targetPort": 2181
      },
      {
        "name": "followers",
        "port": 2888,
        "targetPort": 2888
      },
      {
        "name": "election",
        "port": 3888,
```

```
        "targetPort": 3888
      }
    ],
    "selector": {
      "server-id": "1"
    }
  }
}
```

Zookeeper cluster

---

If you want get a zookeeper cluster with 5 nodes, you can write zookeeper-rc-2/3/4/5.json and zookeeper-svc-2/3/4/5.json files as described above and use kubectl command to deploy them into kubernetes cluster.

Read Kubernetes in production online: https://riptutorial.com/kubernetes/topic/9153/kubernetes-in-production

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with kubernetes | cledoux, Community, Dimitris, idvoretskyi, jayantS, jkantihub, mohan08p, Ogre Psalm33, pagid, PumpkinSeed, webdizz |
| 2 | Calling Kubernetes API | litanhua, Rush |
| 3 | Kubernetes in production | litanhua |