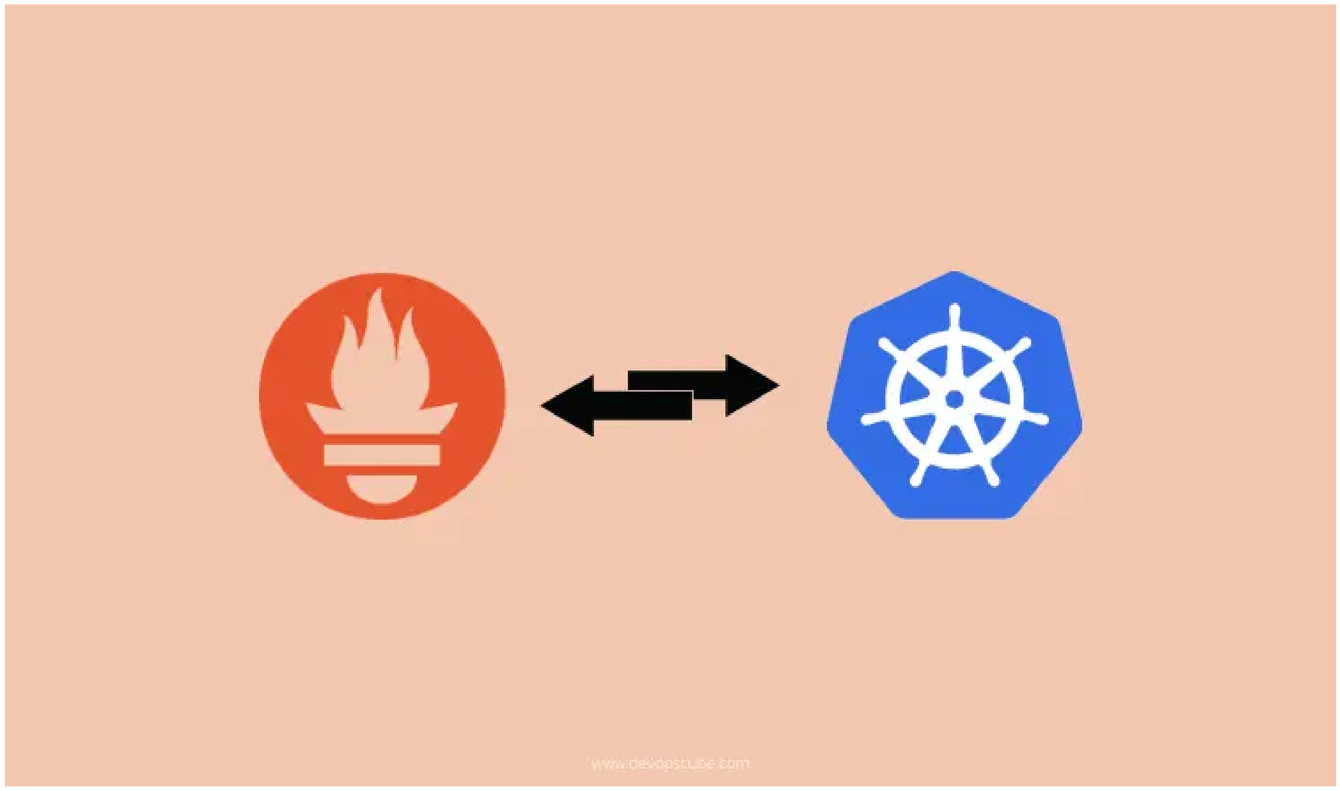
D — DEVOPS

How to Setup Prometheus Monitoring On Kubernetes Cluster

by **Bibin Wilson** · April 7, 2021



This article will guide you through setting up Prometheus on a Kubernetes cluster for monitoring the Kubernetes cluster. This setup collects node, pods, and services metrics automatically using Prometheus service discovery configurations.

About Prometheus

Prometheus is an open-source monitoring framework. It provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform.

Explaining Prometheus is out of the scope of this article. If you want to know more about Prometheus, You can watch all the Prometheus-related videos from here.

- **Metric Collection:** Prometheus uses the pull model to retrieve metrics over HTTP. There is an option to push metrics to Prometheus using `Pushgateway` for use cases where Prometheus cannot Scrape the metrics. One such example is collecting custom metrics from short-lived kubernetes jobs & Cronjobs

- **Metric Endpoint**: The systems that you want to monitor using Prometheus should expose the metrics on an `/metrics` endpoint. Prometheus uses this endpoint to pull the metrics in regular intervals.

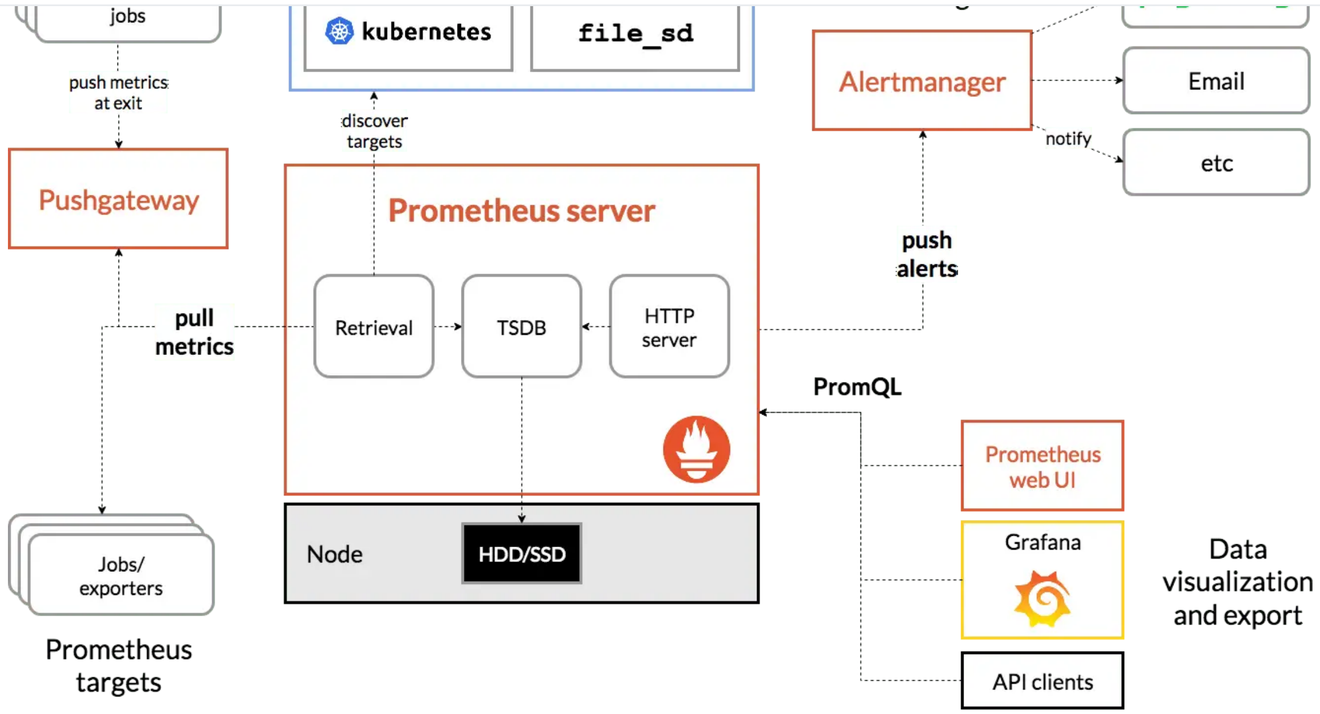
- **PromQL:** Prometheus comes with `PromQL`, a very flexible query language that can be used to query the metrics in the Prometheus dashboard. Also, the PromQL query will be used by Prometheus UI and Grafana to visualize metrics.

- **Prometheus Exporters:** Exporters are libraries which converts existing metric from third-party apps to Prometheus metrics format. There are many official and community Prometheus exporters . One example is, Prometheus node exporter. It exposes all Linux system-level metrics in Prometheus format.

- **TSDB** (time-series database): Prometheus uses TSDB for storing all the data. By default, all the data gets stored locally. However, there are options to integrate remote storage for Prometheus TSDB.

Prometheus Architecture

Here is the high-level architecture of Prometheus.



source: prometheus.io

If you would like to install Prometheus on a Linux VM, please see the Prometheus on Linux guide.

Prometheus Monitoring Setup on Kubernetes

I assume that you have a kubernetes cluster up and running with kubectl setup on your workstation.

> **Note:** If you don't have a kubernetes setup, you can set up a cluster on google cloud by following this article.

Latest Prometheus is available as a docker image in its official docker hub account. We will use that image for the setup.

Connect to your Kubernetes cluster and make sure you have admin privileges to create cluster roles.

> **Only for GKE:** If you are using Google cloud GKE, you need to run the following commands as you need privileges to create cluster roles for this Prometheus setup.

```
ACCOUNT=$(gcloud info --format='value(config.account)')
kubectl create clusterrolebinding owner-cluster-admin-binding \
    --clusterrole cluster-admin \
    --user $ACCOUNT
```

Prometheus Kubernetes Manifest Files

All the configuration files I mentioned in this guide are **hosted on Github**. You can clone the repo using the following command.

```
git clone https://github.com/bibinwilson/kubernetes-prometheus
```

Thanks to James for contributing to this repo. Please don't hesitate to contribute to the repo for adding features.

You can use the Github repo config files or create the files on the go for a better understanding, as mentioned in the steps.

Let's get started with the setup.

Create a Namespace & ClusterRole

First, we will create a Kubernetes namespace for all our monitoring components. If you don't create a dedicated namespace, all the Prometheus kubernetes deployment objects get deployed on the default namespace.

Execute the following command to create a new **namespace named monitoring**.

```
kubectl create namespace monitoring
```

`access` to required API groups and bind the policy to the `monitoring` namespace.

Step 1: Create a file named `clusterRole.yaml` and copy the following RBAC role.

> In the role, given below, you can see that we have added `get`, `list`, and `watch` permissions to nodes, services endpoints, pods, and ingresses. The role binding is bound to the monitoring namespace. If you have any use case to retrieve metrics from any other object, you need to add that in this cluster role.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring
```

Step 2: Create the role using the following command.

```
kubectl create -f clusterRole.yaml
```

Prometheus Configurations

All configurations for Prometheus are part of `prometheus.yaml` file and all the alert rules for Alertmanager are configured in `prometheus.rules` .

1. `prometheus.yaml` : This is the main Prometheus configuration which holds all the scrape configs, service discovery details, storage locations, data retention configs, etc)
2. `prometheus.rules` : This file contains all the Prometheus alerting rules

By externalizing Prometheus configs to a Kubernetes config map, you don't have to build the Prometheus image whenever you need to add or remove a configuration. You need to update the config map and restart the Prometheus pods to apply the new configuration.

The config map with all the Prometheus scrape config and alerting rules gets mounted to the Prometheus container in `/etc/prometheus` location as `prometheus.yaml` and `prometheus.rules` files.

Step 1: Create a file called `config-map.yaml` and copy the file contents from this link –> Prometheus Config File.

Step 2: Execute the following command to create the config map in Kubernetes.

```
kubectl create -f config-map.yaml
```

It creates two files inside the container.

> **Note:** In Prometheus terms, the config for collecting metrics from a collection of endpoints is called a `job` .

The `prometheus.yaml` contains all the configurations to discover pods and services running in the Kubernetes cluster dynamically. We have the following scrape jobs in our Prometheus scrape configuration.

1. `kubernetes-apiservers` : It gets all the metrics from the API servers.

3. `kubernetes-pods`: All the pod metrics get discovered if the pod metadata is annotated with `prometheus.io/scrape` and `prometheus.io/port` annotations.

4. `kubernetes-cadvisor`: Collects all cAdvisor metrics.

5. `kubernetes-service-endpoints`: All the Service endpoints are scrapped if the service metadata is annotated with prometheus.io/scrape and prometheus.io/port annotations. It can be used for black-box monitoring.

`prometheus.rules` contains all the alert rules for sending alerts to the Alertmanager.

Create a Prometheus Deployment

Step 1: Create a file named `prometheus-deployment.yaml` and copy the following contents onto the file. In this configuration, we are mounting the Prometheus config map as a file inside `/etc/prometheus` as explained in the previous section.

> **Note:** This deployment uses the latest official Prometheus image from the docker hub. Also, we are not using any persistent storage volumes for Prometheus storage as it is a basic setup. When setting up Prometheus for production uses cases, make sure you add persistent storage to the deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
  labels:
    app: prometheus-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
```

```
    containers:
      - name: prometheus
        image: prom/prometheus
        args:
          - "--storage.tsdb.retention.time=12h"
          - "--config.file=/etc/prometheus/prometheus.yml"
          - "--storage.tsdb.path=/prometheus/"
        ports:
          - containerPort: 9090
        resources:
          requests:
            cpu: 500m
            memory: 500M
          limits:
            cpu: 1
            memory: 1Gi
        volumeMounts:
          - name: prometheus-config-volume
            mountPath: /etc/prometheus/
          - name: prometheus-storage-volume
            mountPath: /prometheus/
    volumes:
      - name: prometheus-config-volume
        configMap:
          defaultMode: 420
          name: prometheus-server-conf

      - name: prometheus-storage-volume
        emptyDir: {}
```

You Might Like: Kubernetes Deployment Tutorial For Beginners

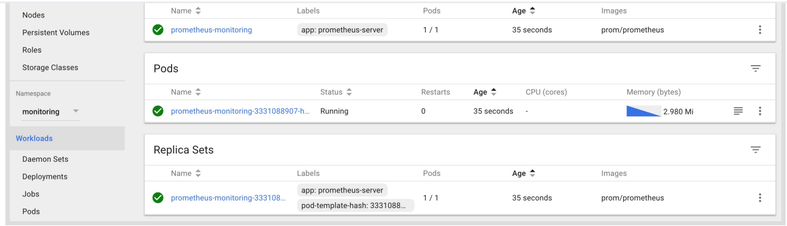
Step 2: Create a deployment on monitoring namespace using the above file.

```
kubectl create  -f prometheus-deployment.yaml
```

Step 3: You can check the created deployment using the following command.

```
kubectl get deployments --namespace=monitoring
```

You can also get details from the kubernetes dashboard like shown below.



Setting Up Kube State Metrics

Kube state metrics service will provide many metrics which is not available by default. Please make sure you deploy Kube state metrics to monitor all your kubernetes API objects like `deployments` , `pods` , `jobs` , `cronjobs` etc..

Please follow this article to setup Kube state metrics on kubernetes ==> How To Setup Kube State Metrics on Kubernetes

Connecting To Prometheus Dashboard

You can view the deployed Prometheus dashboard in three different ways.

1. Using Kubectl port forwarding
2. Exposing the Prometheus deployment as a service with NodePort or a Load Balancer.
3. Adding an Ingress object if you have an Ingress controller deployed.

Let's have a look at all three options.

Using Kubectl port forwarding

Using kubectl port forwarding, you can access a pod from your local workstation using a selected port on your localhost. This method is primarily used for debugging purposes.

Step 1: First, get the Prometheus pod name.

```
kubectl get pods --namespace=monitoring
```

The output will look like the following.

```
prometheus-monitoring-3331088907-hm5n1   1/1       Running   0          5m
```

Step 2: Execute the following command with your pod name to access Prometheus from localhost port 8080.

> **Note:** Replace prometheus-monitoring-3331088907-hm5n1 with your pod name.

```
kubectl port-forward prometheus-monitoring-3331088907-hm5n1 8080:9090 -n
monitoring
```

Step 3: Now, if you access `http://localhost:8080` on your browser, you will get the Prometheus home page.

Exposing Prometheus as a Service [NodePort & LoadBalancer]

To access the Prometheus dashboard over a `IP` or a `DNS` name, you need to expose it as Kubernetes service.

Step 1: Create a file named `prometheus-service.yaml` and copy the following contents. We will expose Prometheus on all kubernetes node IP's on port `30000` .

> **Note:** If you are on AWS, Azure, or Google Cloud, You can use Loadbalancer type, which will create a load balancer and automatically points it to the Kubernetes service endpoint.

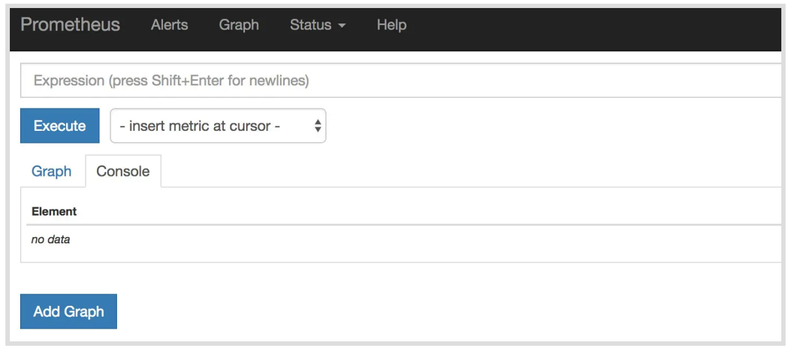
```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '9090'
spec:
  selector:
    app: prometheus-server
  type: NodePort
  ports:
    - port: 8080
```

The `annotations` in the above service YAML makes sure that the service endpoint is scrapped by Prometheus. The `prometheus.io/port` should always be the target port mentioned in service YAML

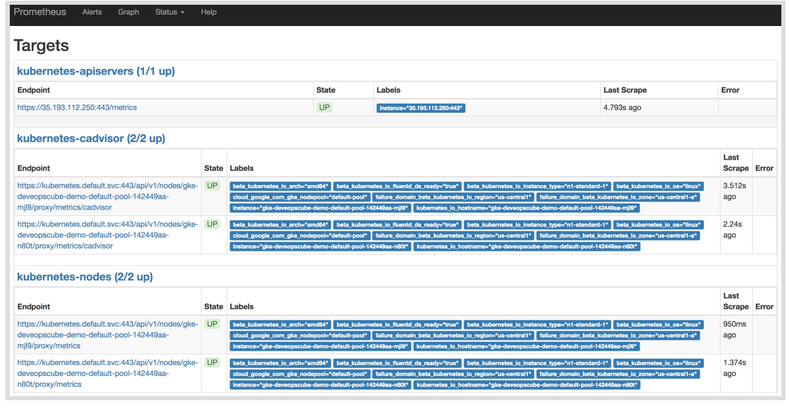
Step 2: Create the service using the following command.

```
kubectl create -f prometheus-service.yaml --namespace=monitoring
```

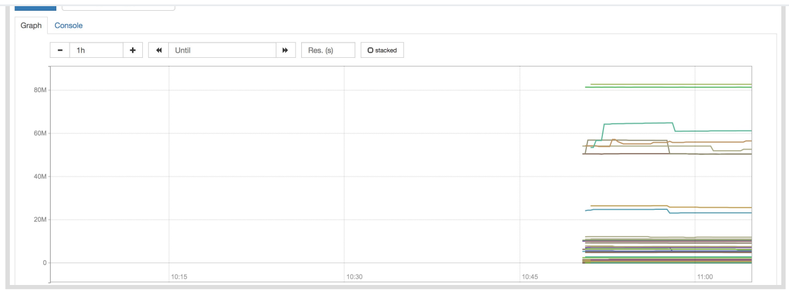
Step 3: Once created, you can access the Prometheus dashboard using any of the Kubernetes nodes IP on port 30000. If you are on the cloud, make sure you have the right firewall rules to access port 30000 from your workstation.



Step 4: Now if you browse to `status --> Targets` , you will see all the Kubernetes endpoints connected to Prometheus automatically using service discovery as shown below.



Step 5: You can head over the homepage and select the metrics you need from the drop-down and get the graph for the time range you mention. An example graph for container memory utilization is shown below.



Exposing Prometheus Using Ingress

If you have an existing ingress controller setup, you can create an ingress object to route the Prometheus DNS to the Prometheus backend service.

Also, you can add SSL for Prometheus in the ingress layer.

Here is a sample ingress object. Please refer to this GitHub link for sample ingress object with SSL

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: prometheus-ui
  namespace: monitoring
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  # Use the host you used in your kubernetes Ingress Configurations
  - host: prometheus.example.com
    http:
      paths:
      - backend:
          serviceName: prometheus-service
          servicePort: 8080
```

Setting Up Alertmanager

Alertmanager handles all the alerting mechanism for Prometheus metrics. There are many integrations available to reveice alerts from the Alertmanager (Slack, email, API endpoints etc)

I have covered the Alert Manager setup in a separate article. Please follow ==> Alert Manager Setup on Kubernetes

Setting Up Grafana

Using Grafana you can create dashboards from Prometheus metrics to monitor the kubernetes cluster.

import it and modify it as per your needs.

Please follow this article for the Grafana setup ==> How To Setup Grafana On Kubernetes

Setting Up Node Exporter

Node Exporter will provide all the Linux system-level metrics of all Kubernetes nodes.

I have written a separate step-by-step guide on node-exporter daemonset deployment. Please follow Setting up Node Exporter on Kubernetes

The scrape config for node-exporter is part of the Prometheus config map. Once you deploy the node-exporter, you should see node-exporter targets and metrics in Prometheus.

Conclusion

In this article, I have covered the setup of important monitoring components for Kubernetes.

For production setup, there are more configuration and parameters need to be considered for scaling and storage. It all depends on your environment and data volume.

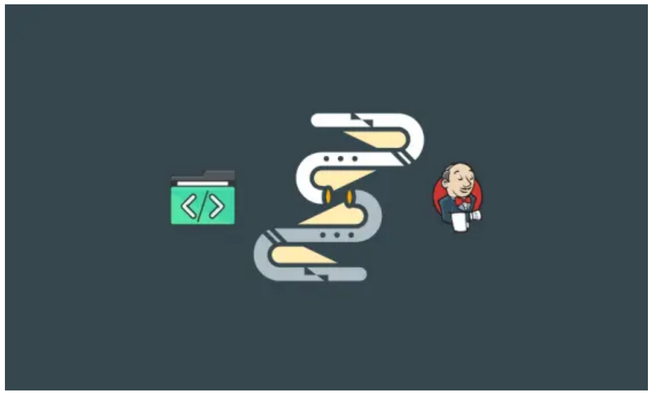
Let me know what you think about the Prometheus monitoring setup by leaving a comment.

TAGS: Monitoring Prometheus

Bibin Wilson

An author, blogger and DevOps practitioner. In spare time, he loves to try out the latest open source technologies. He works as an Associate Technical Architect

YOU MAY ALSO LIKE

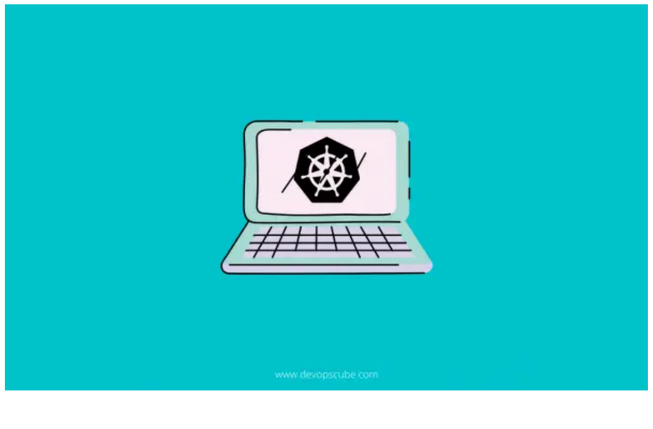


D — DEVOPS

Jenkins Pipeline Tutorial For Beginners

by **Bibin Wilson** · April 11, 2020

Jenkins pipeline as code is a concept of defining Jenkins build pipeline in Jenkins DSL/Groovy format. This article...



K — KUBERNETES

How to Setup Kubernetes Cluster on Vagrant VMs

by **Bibin Wilson** · May 19, 2021

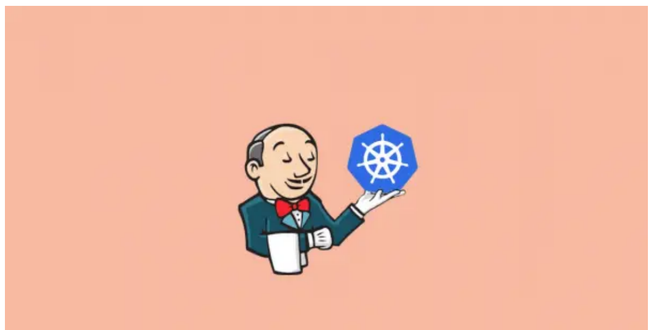
In this post, I have covered the step-by-step guide to setup Kubernetes cluster on Vagrant. It is a...

D — DISTRIBUTED SYSTEMS

How to Setup and Configure Consul Agent On Client Mode

by **devopscube** · April 21, 2019

In our last consul post, we have explained the steps to setup up a multi-node consul cluster which...



C — CI/CD

How To Setup Jenkins On Kubernetes Cluster – Beginners Guide

by **Bibin Wilson** · May 9, 2021

Hosting Jenkins on a Kubernetes cluster is beneficial for Kubernetes-based

D — DEVOPS

How To Backup Jenkins Data and Configurations

by **Bibin Wilson** · June 14, 2021

It is very important to have Jenkins backup with its data and configurations. It includes job configs, builds...

D — DEVOPS

Release Management Explained: Dev To Prod Deployment Process

by **Nivedha Varadharajan** · April 29, 2020

If you are into DevOps it is very important to understand release management. There is no single template...