

Be part of a better internet. [Get 20% off membership for a limited time](#)

# How to Install a Kubernetes Cluster on RHEL 8

Prerequisites



Srin · [Follow](#)

6 min read · Sep 7, 2020



Listen



Share

... More

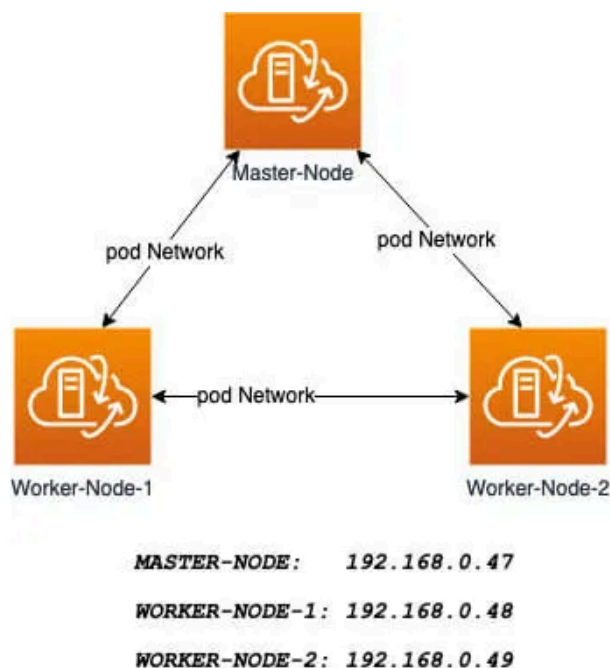
1. Three servers running RHEL8 –1 Master Node and 2 Worker Nodes.
2. It is recommended that your nodes should have at least 2 CPUs with 2GB RAM or more per machine. This is not a strict requirement but is largely driven by the needs of the application you intend to run.
3. Internet connectivity on all your nodes. We will be fetching **Kubernetes** and **docker** packages from the repository. Equally, you will need to make sure that the DNF package manager is installed by default and can fetch packages remotely.
4. All your nodes should also be able to connect to one another, either on a private or public network, whichever is available.
5. You will also need access to an account with sudo or root privileges. In this tutorial, I will be using my root account.

## Logical Architecture

Our installation is designed to have the **Master-Node** controlling the **Worker Nodes**. At the end of this installation, our logical architecture will look something like this.

**Master Node** — This machine generally acts as the control plane and runs the cluster database and the API server (which the kubectl CLI communicates with).

Our 3-node **Kubernetes Cluster** will look something like this:



## Installation of Kubernetes Cluster on Master-Node

For Kubernetes to work, you will need a containerization engine. As mentioned, we will be using Docker-CE.

The following institutions will be performed on **RHEL 8 Master-Node**.

### Step 1: Install Docker-CE on RHEL 8

You will need to add the **Docker** repository first as it is no longer in the default package list using the following **dnf config-manager** command.

```
dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo
```

Also install **containerd.io** package which is available as a daemon that manages the complete container lifecycle of its host system, from image transfer and storage to container execution and supervision to low-level storage to network attachments and beyond.

```
dnf install
https://download.docker.com/linux/centos/7/x86_64/stable/Packages/containerd.io-1.2.6-3.3.el7.x86_64.rpm
```

Now install the latest version of a **docker-ce** package.

```
systemctl enable docker  
systemctl start docker
```

## Step 2: Install Kubernetes (Kubeadm) on RHEL 8

Next, you will need to add **Kubernetes** repositories manually as they do not come installed by default on **RHEL 8**.

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo  
[kubernetes]  
name=Kubernetes  
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64  
enabled=1  
gpgcheck=1  
repo_gpgcheck=1  
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
EOF
```

**Kubeadm** helps you bootstrap a minimum viable Kubernetes cluster that conforms to best practices. With **kubeadm**, your cluster should pass the Kubernetes Conformance tests.

**Kubeadm** also supports other cluster lifecycle functions, such as upgrades, downgrade, and managing bootstrap tokens. Kubeadm is also integration-friendly with other orchestration tools like **Ansible** and **Terraform**.

With the package repo now ready, you can go ahead and install **kubeadm** package.

```
dnf install kubeadm -y
```

When the installation completes successfully, enable and start the service.

```
systemctl enable kubelet  
systemctl start kubelet
```

## Step 3: Create a control-plane Master with kubeadm

The Kubernetes master which acts as the **control plane** for the cluster runs a few critical services necessary for the cluster. As such, the initialization process will do a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors. **kubeadm init** then downloads and installs the cluster control plane components.

Now it's time to initialize Kubernetes master, but before that, you must disable swap in order to run “**kubeadm init**” command.

```
# swapoff -a
```

Initializing **Kubernetes** master is a completely automated process that is controlled by the “**kubeadm init**” command as shown.

```
# kubeadm init
```

```
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 31.501735 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-X.Y" in namespace kube-system with the configuration for the kubelets in the cluster
[patchnode] Uploading the CRI Socket information "/var/run/dockerhim.sock" to the Node API object "kubeadm-cp" as an annotation
[mark-control-plane] Marking the node kubeadm-cp as control-plane by adding the label "node-role.kubernetes.io/master="
[mark-control-plane] Marking the node kubeadm-cp as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: <token>
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a Pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node as root:

kubeadm join 192.168.0.47:6443 --token nu06lu.xrsux0ss0ixtnms5 \
--discovery-token-ca-cert-hash sha256:f996ea35r4353d342fdea2997a1cf8caeddaafd6d4360d606dbc82314683478hjm783
```

#### Initialize Kubernetes Master

Next, copy the following command and store it somewhere, as we required to run this command on the worker nodes later.

```
kubeadm join 192.168.0.47:6443 --token nu06lu.xrsux0ss0ixtnms5 \ --discovery-
token-ca-cert-hash
ha256:f996ea35r4353d342fdea2997a1cf8caeddaafd6d4360d606dbc82314683478hjm7
```

**Tip:** Sometimes the above command might throw errors about the arguments passed, so to avoid errors, you need to remove the ‘\’ character and your final command will look like this.

```
kubeadm join 192.168.0.47:6443 --token nu06lu.xrsux0ss0ixtnms5 -discovery
token-ca-cert-hash
```

sha256:f996ea35r4353d342fdea2997a1cf8caeddaafd6d4360d606dbc82314683478hjm7

Once **Kubernetes** initialized successfully, you must enable your user to start using the cluster. In our scenario, we will be using the root user. You can also start the cluster using sudo user as shown.

To use **root**, run

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

Now confirm that the **kubectl** command is activated.

```
# kubectl get nodes
```

```
[root@master-node home]# kubectl get nodes
NAME             STATUS    ROLES    AGE   VERSION
master-node      NotReady  master   6m27s v1.17.0
[root@master-node home]#
```

Check Status of Nodes

At this moment, you will see the status of the master-node is '**NotReady**'. This is because we are yet to deploy the pod network to the cluster.

The pod Network is the overlay network for the cluster, that is deployed on top of the present node network. It is designed to allow connectivity across the pod.

### Step 5: Setup Your Pod Network

Deploying the network cluster is a highly flexible process depending on your needs and there are many options available. Since we want to keep our installation as simple as possible, we will use **Weavenet** plugin which does not require any configuration or extra code and it provides one IP address per pod which is great for us. If you want to see more options, please check [here](#).

These commands will be important to get the pod network setup.

```
export kubever=$(kubectl version | base64 | tr -d '\n')
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"
```

Now if you check the status of your master-node, it should be **'Ready'**.

```
kubectl get nodes
```

```
[root@master-node home]# kubectl get nodes
NAME             STATUS    ROLES    AGE   VERSION
master-node      Ready    master   23m   v1.17.0
[root@master-node home]#
```

Check Status of Master Nodes

Next, we add the **worker nodes** to the cluster.

## Adding Worker Nodes to Kubernetes Cluster

The following instructions will be performed on each **worker node** when joining the Kubernetes cluster.

### Step 1: Prepare Hostname, Firewall, and SELinux

First set the hostname on your **worker-node-1** and **worker-node-2**, and then add the host entries to the **/etc/hosts** file.

```
hostnamectl set-hostname 'node-1'

cat <<EOF>> /etc/hosts
192.168.0.47 master-node
192.168.0.48 node-1 worker-node-1
192.168.0.49 node-2 worker-node-2
EOF
```

Next, ping your **master node** from your **worker nodes** to confirm that your updated host file is working fine using the [ping command](#).

```
# 192.168.0.47
```

Next, disable **SELinux** and update your firewall rules.

```
# setenforce 0
# sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g'
/etc/sysconfig/selinux
# firewall-cmd --permanent --add-port=6783/tcp
# firewall-cmd --permanent --add-port=10250/tcp
# firewall-cmd --permanent --add-port=10255/tcp
```

```
# firewall-cmd --permanent --add-port=30000-32767/tcp
# firewall-cmd --reload
# echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
```

## Step 2: Setup Docker-CE and Kubernetes Repo

Add the Docker repository first using DNF config-manager.

```
# dnf config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo
```

Next, add the **containerd.io** package.

```
# dnf install
https://download.docker.com/linux/centos/7/x86_64/stable/Packages/containerd.io
-1.2.6-3.3.el7.x86_64.rpm
```

With these two packages installed, install the latest version of **docker-ce**.

```
# dnf install docker-ce
```

Enable and start the docker service.

```
# systemctl enable docker
# systemctl start docker
```

You will need to add **Kubernetes** repositories manually as they do not come pre-installed on **REDHAT 8**.

```
# cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

## Step 3: Install Kubeadm on REDHAT 8

With the package repo now ready, you can go ahead and install **kubeadm**.

```
# dnf install kubeadm -y
```

Start and enable the service.

```
# systemctl enable kubelet  
# systemctl start kubelet
```

#### Step 4: Join the Worker Node to the Kubernetes Cluster

We now require the token that **kubeadm init** generated, to join the cluster. You can copy and paste it to your **node-1** and **node-2** if you had copied it somewhere.

```
# kubeadm join 192.168.0.47:6443 --token nu06lu.xrsux0ss0ixtnms5 --discovery-  
token-ca-cert-hash  
sha256:f996ea35r4353d342fdea2997a1cf8caeddaafd6d4360d606dbc82314683478hjm78
```

As suggested on the last line, go back to your **master-node** and verify if worker **node-1** and **worker node-2** have joined the cluster using the following command.

```
# kubectl get nodes
```

```
[root@master-node home]# kubectl get nodes  
NAME           STATUS    ROLES    AGE   VERSION  
master-node    Ready    master   79m   v1.17.0  
node-1         Ready    <none>   18m   v1.17.0  
node-2         Ready    <none>   2m8s  v1.17.0  
[root@master-node home]#
```

Check All Nodes Status in Kubernetes Cluster

If all the steps run successfully, then, you should see **node-1** and **node-2** in ready status on the **master-node**. At this point, you have now successfully deployed a Kubernetes cluster on **RHEL 8**.





Follow




## Written by Srin

16 Followers

### More from Srin



 Srin

## Install and Configure git in Linux EC2 Instance

Install Server Updates

Sep 10, 2020  1



```
python-ecdsa python-enum34 python-httplib2 python-idna
python-aml sshpass

python-doc python-cryptography-vectors python-enum34-doc python-cryptography python-ecdsa python-enum34 python-httplib2 python-aml sshpass
python-raded.
```



Srin

## Setting up Windows Host using Ansible

1. Install ansible

Oct 15, 2020 🖱 2



```
node_exporter.service; disabled; vendor preset: enabled)
2020-08-26 23:41:17 UTC; 49s ago
```

```
ce
ter
```

```
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:112
[16196]: level=info ts=2020-08-26T23:41:17.174Z caller=node_exporter.go:191
```



Srin

## Monitor Linux Server using Prometheus Node Exporter

Prerequisites

Aug 27, 2020 🖱 3



2 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 4 GiB memory, EBS only)

Type ▾	vCPUs ⓘ ▾	Memory (GiB) ▾	Instance Storage (GB) ⓘ ▾	EBS-Optimized Available ⓘ ▾	N
t2.nano	1	0.5	EBS only	-	
t2.micro Free tier eligible	1	1	EBS only	-	
t2.small	1	2	EBS only	-	
t2.medium	2	4	EBS only	-	
t2.large	2	8	EBS only	-	

[Cancel](#)[Previous](#)[Review and Launch](#)

Srin

## How to Install Prometheus on Kubernetes Cluster | AWS | Ubuntu 16.04|

Step1: Create AMI instance of type Ubuntu Server 16.04 LTS using your AWS login


Aug 27, 2020

[See all from Srin](#)

## Recommended from Medium



## Install Kubernetes Cluster (kubeadm Setup) on Ubuntu 24.04 LTS

 Subham Pradhan

### How to Install Kubernetes Cluster (kubeadm Setup) on Ubuntu 24.04 LTS (Step-by-Step Guide)

This document provides a step-by-step guide to setting up a Kubernetes cluster using kubeadm on multiple nodes. Kubernetes is an...

Apr 4  9  2



**Amazon.com**

Seattle, WA

*Software Development Engineer*

Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

## Projects

### NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

### HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

## The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.



Jun 1



16.2K



261

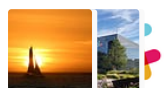


## Lists



### Staff Picks

706 stories · 1199 saves



### Stories to Help You Level-Up at Work

19 stories · 728 saves



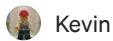
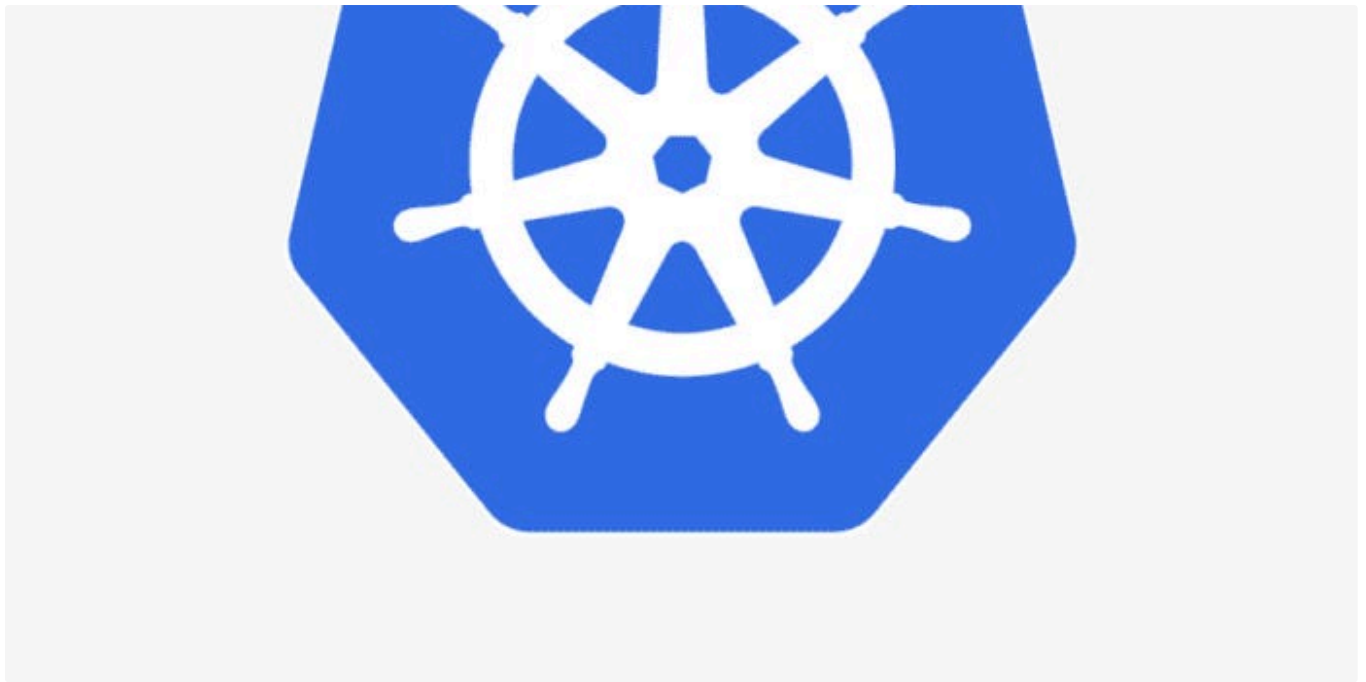
### Self-Improvement 101

20 stories · 2476 saves



### Productivity 101

20 stories · 2164 saves

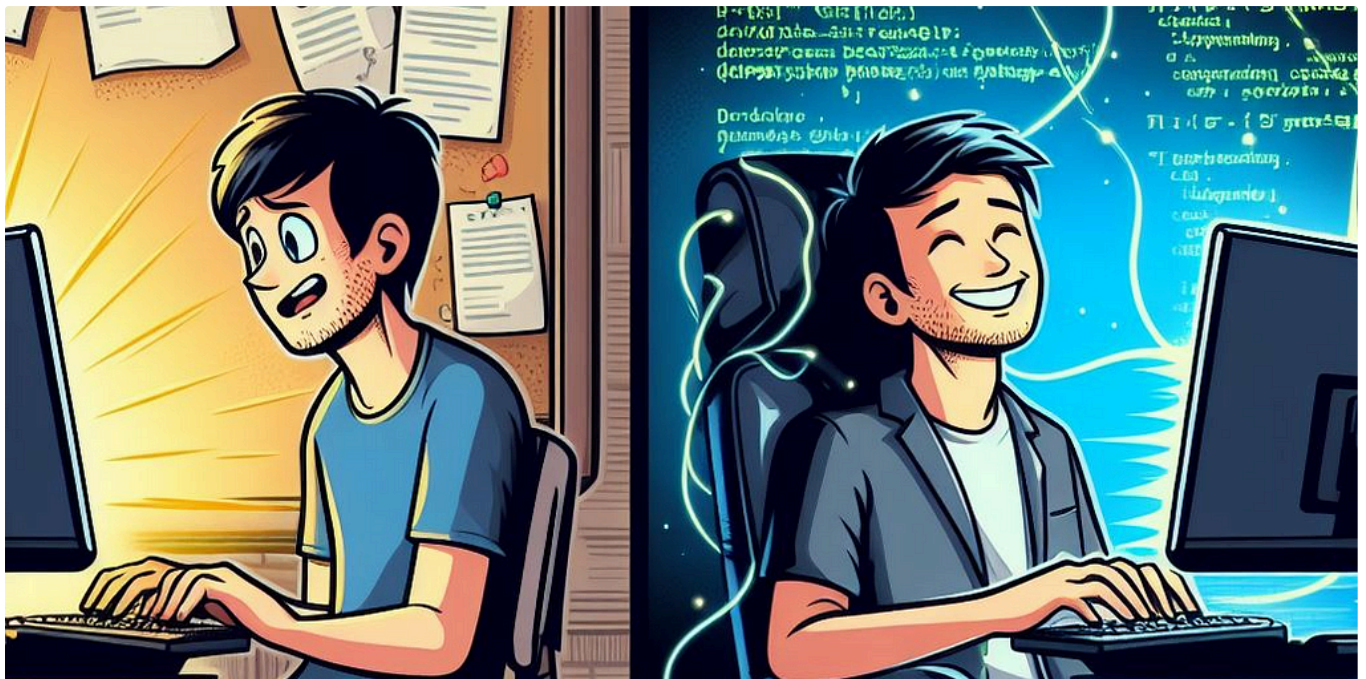


Kevin

## Kubectl Command Cheat Sheet

A Comprehensive Kubectl Command Cheat Sheet for Streamlined Operations

★ Mar 1 🖱️ 23



Abhay Parashar in The Pythoneers


## 17 Mindblowing Python Automation Scripts I Use Everyday


Scripts That Increased My Productivity and Performance

★ Jul 29 🖱️ 6K 💬 45



# Count(\*) vs Count(1)



 Vishal Barvaliya in Data Engineer


## Count(\*) vs Count(1) in SQL.

If you've spent any time writing SQL queries, you've probably seen both `COUNT(*)` and `COUNT(1)` used to count rows in a table. But what's...

★ Mar 9 🖱 843 💬 20





 Serxan Hamzayev in JavaToDev

## Advanced Query Techniques with Elasticsearch in Spring Boot

Elasticsearch is a powerful, distributed search and analytics engine that allows for the quick and efficient searching of complex data...

★ Feb 18 🖱 14



See more recommendations