native labs

Infrastructures
& Security solutions

Dear readers,

Hey there! How have you been? We know it has been a while. We were working on how to make Native LABs community and its publication better for you and so we decided to make a few changes. One of the major changes is that we publish this monthly instead of weekly like we used to. We missed you and we hope that you did too and that you had a swell time reading through last month's curation.
As usual, we bring the top news and stories published on Native LABs's channels—articles, tutorials, tips, that were most read and interacted with.

## Summary:


1. Top 10 Kubernetes CI/CD tools 3
2. Building Kubernetes over three nodes Centos/RHEL 7 6
3. Writing a fast IP/TCP scanner using Golang 9
4. Postfix High Availability servers as like as Yahoo 12

# 1. Top 10 Kubernetes CI/CD tools

A CI/CD pipeline in a Kubernetes based environment is usually composed of 4 main components:

Version control system, CI system, Docker Registry, The Kubernetes cluster. The coordination and automation of the above pieces make for seamless and continuous software delivery. Here is the list of the top 10 Kubernetes CI/CD tools. Kubernetes, also called Kube, is an open-source platform used for container orchestration, which is managing the lifecycle of containers in dynamic environments.

Kubernetes is portable, extensible, and scalable. It has become quite common to club Kubernetes with CI/CD (Continuous Integration/Continuous Delivery) tools. When CI/CD tools are used to deploy cloud-native applications, it streamlines the development as well as the deployment phases. You can make cloud-native applications much more robust if you integrate their development with CI/CD approach.

Continuous delivery in Kubernetes is more efficient compared to the traditional VM deployments. You need not bring down an entire application to update or make changes. The coordination and automation of the above pieces make for seamless and continuous software delivery. Here is the list of the top 10 Kubernetes CI/CD tools:

### 1) Helm

Helm is one of the best-known package managers for Kubernetes. It uses “charts,” which is basically a definition of a Kubernetes package and any other dependencies that you need for your app. When you call a chart from the command line, Helm creates YAML files for Kubernetes deployment and then adds them to the cluster. Helm is open-source, which means that you can download, change, and use the charts for your own organization.

The biggest advantage of Helm is that it makes the deployment of complex applications more portable. Helm also supports automatic rollbacks and is easier for developers to understand. The disadvantage of Helm is that it is difficult to set up and maintain.

### 2) Ksonnet and Jsonnet

Ksonnet (built on the JSON templating language Jsonnet) is a configuration management tool. It provides a way to package your Kubernetes resources, which can be later on used to create the required configuration files for your deployments. Ksonnet is the command-line interface, while Jsonnet is the data templating language used to describe your application.

This tool's advantage is that developers who are familiar with JSON can easily use Jsonnet to deploy their applications. However, there are differences between just using JSON and working with Jsonnet, which a developer would, of course, need to learn.

**3) Draft**

Draft is a tool for creating cloud-based applications on Kubernetes. It is a build and deployment tool developed by Microsoft. Draft can be used to generate Docker images after your code has gone through Continuous Integration. You can also use it to create Helm charts which generate YAML files used to deploy applications through Kubernetes. This tool's advantage is that you can combine it with Helm to package applications and deploy them. The disadvantage is that it requires a considerable amount of configuration.

**4) Jenkins X**

Jenkins X is one of the most popular and powerful CI tools used for Kubernetes deployment. It is an open-source automation tool with built-in plugins for CI purposes. It is written in Java.

You can use Jenkins to continuously build and test your software projects so that it becomes easier for you to make changes to the project. You can use this tool to continuously deliver your project through the integration of a large number of testing and deployment technologies. Even though it is a powerful tool, Jenkins can be complex and prone to errors.

**5) CircleCI**

CircleCI is another continuous integration and delivery tool. It is a cloud-based tool that includes an API (Application Program Interface) for automatic Kubernetes deployment. Since it is cloud-based, you do not need a dedicated server for CircleCI.

The advantage of CircleCI is that it uses a number of testing methods like unit testing, integration testing, and functional tests to test any change to the code before deployment. The disadvantage of the tool is that it lacks all the pieces to make it a complete CD pipeline.

**6) Travis**

Travis is a commercial CI tool, unlike Jenkins. You can use the tool to sign up, link your repository, build, as well as test your applications. You can also integrate the tool with other common cloud repositories like Bitbucket and GitHub. Travis is a cloud-based tool that cuts out the need for a dedicated server. The tool allows you test on various machines running different operating systems.

Travis is free for open-source projects, but you will need to purchase an enterprise plan costing about $69 a month for commercial projects.

**7) Gitlab**

Gitlab is a web-based tool with CI/CD pipeline features. It also has its own repository containing wikis, code review, issue tracking, besides a CI/CD deployment tool. Gitlab is an open-source platform that can effortlessly handle nearly 25,000 users on a single server. It also comes with in-built auto-deployment Kubernetes components and also can support Helm charts.

### 8) Weave Cloud

Weave Cloud is a CD tool that gives you a fast way to monitor and manage your Docker containers. It also gives a quick way to set up your CI/CD pipeline and Kubernetes cluster. It allows you to deploy your apps with faster rollouts, updates, and rollbacks. This tool works by using Git as a single source of truth for declarative infrastructure and applications. The downside is that it needs quite a bit of configuration to start working properly.

### 9) Spinnaker

Spinnaker is an open-source tool developed by Netflix. It manages pipelines and deployments and can also support Helm charts. It is an open-source, multi-cloud tool that provides continuous delivery with great efficiency. The disadvantage is that this tool was originally built to support VMs and not Kubernetes, and hence can be a little complicated to set up.

### 10) Codefresh

Codefresh is a CD/CD pipeline tool that also supports Helm charts. It allows you to use your own CI and image repository. It helps you build a simple yet powerful CI/CD pipeline. It comes with an extensive set of plugins that help you integrate with your favorite tools. The downside is that third-party tools are set up with their Graphical User Interface, which adds more complications to your pipeline. Codefresh is a commercial tool and its pricing starts from $34 a month.

### Conclusion

In recent times developers have become more familiar with Continuous Integration and Continuous Delivery. You cannot think of creating new software without having a proper CI/CD pipeline. Kubernetes is fast gaining popularity, and all the tools are being upgraded to integrate with Kubernetes. We have tried to compile a list of the most popular Kubernetes CI/CD tools for you in the article above. What are the tools you use to create and manage your CI/CD pipelines? Leave them in the comments section.

# 2. Building Kubernetes over three nodes Centos/RHEL 7

In the next section a mini how to install Kubernetes in three nodes Centos/RHEL Operating systems.

Kubernetes as explained above is an open source container orchestration solution on server clusters, it also allows deployment automation and scalability, created by Google. Kubernetes works with several containerization technologies, but it is often used with Docker. There are several on the market: docker-swarm the solution offered by Docker and Mesos. In this draft we will focus on the following points:

- **Installation of Kubernetes on a cluster of three Centos 7.4 machines**
- **Methods of exposing services to the internet**
- **High availability service deployment**
- **Service update and rollback**

We will install Kubernetes on an architecture composed of a master server and two nodes under Centos 7.4.

Master server: cygne - Knots: maya, troy

The installation of kubernetes will be done with kubeadm the installation process is the same on all servers: master and nodes only the initialization of the role of the server differs. Place yourself on the master: Create a repository to retrieve the kubernetes components:

```
mezgani@cygne ~ % 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

mezgani@cygne ~ % yum -y install firewalld

mezgani@cygne ~ % firewall-cmd --permanent --add-port=6443/tcp
mezgani@cygne ~ % firewall-cmd --permanent --add-port=2379-2380/tcp
mezgani@cygne ~ % firewall-cmd --permanent --add-port=10250/tcp
mezgani@cygne ~ % firewall-cmd --permanent --add-port=10251/tcp
mezgani@cygne ~ % firewall-cmd --permanent --add-port=10252/tcp
mezgani@cygne ~ % firewall-cmd --permanent --add-port=10255/tcp
mezgani@cygne ~ % firewall-cmd --reload
mezgani@cygne ~ % modprobe br_netfilter
mezgani@cygne ~ % echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

mezgani@cygne ~ % setenforce 0
```

```
mezgani@cygne ~ % sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/
selinux
```

```
mezgani@cygne ~ % yum install kubelet kubectl kubernetes-cni kubeadm docker -y
mezgani@cygne ~ % systemctl restart docker && systemctl enable docker
mezgani@cygne ~ % systemctl restart kubelet && systemctl enable kubelet
mezgani@cygne ~ % kubeadm init --pod-network-cidr=10.244.0.0/16
mezgani@cygne ~ % mkdir -p $HOME/.kube
mezgani@cygne ~ % cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
mezgani@cygne ~ % chown $(id -u):$(id -g) $HOME/.kube/config
mezgani@cygne ~ % kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/
Documentation/kube-flannel.yml
mezgani@cygne ~ % kubectl get nodes
mezgani@cygne ~ % kubectl get pods --all-namespaces
```

On nodes

```
mezgani@maya ~ % setenforce 0
mezgani@maya ~ % sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/
selinux
mezgani@maya ~ % firewall-cmd --permanent --add-port=10250/tcp
mezgani@maya ~ % firewall-cmd --permanent --add-port=10255/tcp
mezgani@maya ~ % firewall-cmd --permanent --add-port=30000-32767/tcp
mezgani@maya ~ % firewall-cmd --permanent --add-port=6783/tcp
mezgani@maya ~ % firewall-cmd --reload
mezgani@maya ~ % modprobe br_netfilter
mezgani@maya ~ % echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
mezgani@maya ~ % 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
```

```
mezgani@maya ~ % yum install kubeadm docker -y
mezgani@troy ~   % yum install kubeadm docker -y
```

Start Docker service on both of nodes :

```
mezgani@maya ~ % systemctl restart docker && systemctl enable docker
mezgani@troy ~   % systemctl restart docker && systemctl enable docker
```

Join worker nodes to master node.

```
kubeadm join --token 04805f.f03cde332a2c52b9 174.138.8.89:6443 --discovery-token-ca-cert-hash
sha256:053a6ba49e2b6f49587a5e08a68ebc80b5b30168633e2a181206a4bb3075d33a

mezgani@maya ~ % kubeadm join --token a3bd48.1bc42347c3b35851 192.168.1.30:6443
mezgani@troy ~   % kubeadm join --token a3bd48.1bc42347c3b35851 192.168.1.30:6443
```

Using ansible, where 192.168.1.30 is the Internet protocol address of cygne:

```
ansible -i hosts 'kube:!cygne' -m shell -a "kubeadm join --token a3bd48.1bc42347c3b35851
192.168.1.30:6443"

mezgani@cygne ~ % kubectl get nodes
NAME STATUS ROLES AGE VERSION
troy    Ready <none> 31s v1.9.0
maya  Ready <none> 44s v1.9.0
cygne Ready master 4m v1.9.0
```

Ok, you've run one of the getting started guides and you have successfully turned up a Kubernetes cluster. Now what? This guide will help you get oriented to Kubernetes and running your first containers on the cluster. From this point onwards, it is assumed that kubectl is on your path from one of the getting started guides. The kubectl run line below will create two nginx pods listening on port 80. It will also create a deployment named my-nginx to ensure that there are always two pods running.

```
mezgani@cygne ~ % kubectl run my-nginx --image=nginx --replicas=2 --port=80
```

Once the pods are created, you can list them to see what is up and running:

```
mezgani@cygne ~ % kubectl get pods
```

You can also see the deployment that was created:

```
mezgani@cygne ~ % kubectl get deployment
```

On some platforms (for example Google Compute Engine) the kubectl command can integrate with your cloud provider to add a public IP address for the pods, to do this run the command below to expose your pods to the internet.

```
mezgani@cygne ~ % kubectl expose deployment my-nginx --port=80 --type=LoadBalancer
```

At this point, you are ready to deploy a service on your Kubernetes cluster. To deploy an NGINX service (and expose the service on port 80), run the following commands (from the master):

```
mezgani@cygne ~ % kubectl run --image=nginx nginx-app --port=80 --env="DOMAIN=cluster"
mezgani@cygne ~ % kubectl expose deployment nginx-app —port=80 —name=nginx-http
```

This should print the service that has been created, and map an external IP address to the service. Where to find this external IP address will depend on the environment you run in. For instance, for Google Compute Engine the external IP address is listed as part of the newly created service and can be retrieved by running

```
mezgani@cygne ~ % kubectl get services
```

In order to access your nginx landing page, you also have to make sure that traffic from external IPs is allowed. Do this by opening a firewall to allow traffic on port 80.

To delete the two replicated containers, delete the deployment:
mezgani@cygne ~ % kubectl delete deployment my-nginx

the best way is to deploy an ingress controller and an elb in front. The elb will land port 80 and 443 and send it back to an ingress controller on a node port, then ingress resources you define will be the backend services, this is a lot to cover quickly and the ingress docs help alot.

Most people will eventually want to use declarative configuration files for creating/modifying their applications. A simplified introduction is given in a different document.

mezgani@cygne ~ % kubectl describe svc/my-nginx
mezgani@cygne ~ % kubectl scale deployment http-svc --replicas=4
mezgani@cygne ~ % kubectl exec haproxy-ingress-1373648123-dqkkc pidof haproxy

This will un-configure the kubernetes cluster.
mezgani@cygne ~ % kubeadm reset

For diagnostic :
mezgani@cygne ~ % kubectl describe node <NotReady node>

SSH'd into the NotReady status node, and run : journalctl -u kubelet --all | tail

We become to the end of the first tutorial. In the fact we are ready now with nginx running on our nodes cluster and exposed to the internet. This is very interesting as first approach to Kubernetes up and running. Choose a light cloud provider as like as DigitalOcean, Civo or maybe Vultr.

## 3. Writing a fast IP/TCP scanner using Golang

One effective way to conceptualize the interaction of TCP ports is by implementing a port scanner. By writing one, you'll observe the steps that occur in a TCP handshake, along with the effects of encountered state changes, which allow you to determine whether a TCP port is available or whether it responds with a closed or filtered state.

Once you've written a basic scanner, you'll write one that's faster. A port scanner may scan several ports by using a single contiguous method; however, this can become time-consuming when your goal is to scan all 65,535 ports. You'll explore how to use concurrency to make an inefficient port scanner more suitable for larger port-scanning tasks.

You'll also be able to apply the concurrency patterns that you'll learn in this section in many other scenarios, both in this part and beyond.

scanner.go : A basic scanner that scans only the port 80 if the host www.nativelabs.ma

```
package main
import (
    "fmt"
```

```
    "net"
)

func main() {
    _, err := net.Dial("tcp", "www.nativelabs.ma:80")
    if err == nil {
    fmt.Println("Connection successful")
    }
}
```

Run this code. You should see Connection successful, provided you have access to the great information superhighway.

Scanning a single port at a time isn't useful, and it certainly isn't efficient. TCP ports range from 1 to 65535; but for testing, let's scan ports 1 to 1024. To do this, you can use a for loop:

```
package main
import (
    "fmt"
    "net"
)

func main() {
    for i := 1; i <= 1024; i++ {
        address := fmt.Sprintf("www.nativelabs.ma:%d", i)
        conn, err := net.Dial("tcp", address)
        if err != nil {
            // port is closed or filtered.
            continue
        }
        conn.Close()
        fmt.Printf("%d open\n", i)
    }
}
```

Compile and execute this code to conduct a light scan against the  target. You should see a couple of open ports.

The previous scanner scanned multiple ports in a single go (pun intended). But your goal now is to scan multiple ports concurrently, which will make your port scanner faster. To do this, you'll harness the power of goroutines. Go will let you create as many goroutines as your system can handle, bound only by available memory.

```
package main
import (
    "fmt"
    "net"
)
func main() {
    for i := 1; i <= 1024; i++ {
    go func(j int) {
        address := fmt.Sprintf("www.nativelabs.ma:%d", j)
        conn, err := net.Dial("tcp", address)
        if err != nil {
            return
        }
        conn.Close() fmt.Printf("%d open\n", j)
        }(i)
    }
}
```

Upon running this code, you should observe the program exiting almost immediately. Below we build a TCP scanner using multi-channels.

```
package main
import (
    "fmt"
    "net"
    "sort"
)

func worker(ports, results chan int) {
    for p := range ports {
        address := fmt.Sprintf("::1:%d", p)

        conn, err := net.Dial("tcp", address)
        if err != nil {
            results <- 0
            continue
        }
        conn.Close()
        results <- p
    }
}

func main() {
    ports := make(chan int, 1024)
    results := make(chan int)
    var openports []int
    for i := 0; i < cap(ports); i++ {
        go worker(ports, results)
```

```
    }

    go func() {
        for i := 0; i < 65535; i++ {
            ports <- i
        }
    }()
    for i := 0; i < 65535; i++ {
        port := <-results
        if port != 0 {
            openports = append(openports, port)
        }
    }
    close(ports)
    close(results)
    sort.Ints(openports)
    for _, port := range openports {
        fmt.Printf("%d open\n", port)
    }
}
```

The worker(ports, results chan int) function has been modified to accept two channels; the remaining logic is mostly the same, except that if the port is closed, you'll send a zero , and if it's open, you'll send the ports. Also, you create a separate channel to communicate the results from the worker to the main thread. You then use a slice to store the results so you can sort them later. Next, you need to send to the workers

in a separate goroutine because the result-gathering loop needs to start before more than 100 items of work can continue.

The result-gathering loop{receives on the results channel 1024 times. If the port doesn't equal 0, it's appended to the slice. After closing the channels, you'll use sort|to sort the slice of open ports. All that's left is to loop over the slice and print the open ports to screen.

There you have it: a highly efficient port scanner. Take some time to play around with the code—specifically, the number of workers. The higher the count, the faster your program should execute. But if you add too many workers, your results could become unreliable. When you're writing tools for others to use, you'll want to use a healthy default value that caters to reliability over speed. However, you should also allow users to provide the number of workers as an option.

## 4. Postfix High Availability servers as like as Yahoo

To build a dedicated server hosting provider with approximately 2,000 servers on your network. We have ran into an issue with unnecessary IP blacklisting due to customer abuse. In order to ensure the security of your network, we have designed a new platform consisting of 8 machines running Debian.

2 x HA Proxy Load Balancers running Keepalived for automatic failover of a virtual IP
4 x Postfix mail servers
2 x Syslog Servers for central logging

It will be great to place a policy based route at your network edge routers taking all outbound port 25 traffic and divert it to a next hop of the load balancer's virtual IP address. The load balancer servers are configured to take all traffic received on port 25 and redirect it to localhost on port 25 using IPTables, so regardless of end destination the user will try to connect to, the session will be intercepted by the load balancers. HAProxy is then configured to round-robin the requests received on port 25 to the 4 Postfix mail servers.

The Postfix mail servers are setup to accept relaying from any IP address on the network, but will only accept traffic coming through the load balancers. The mail servers then forward the email to an offsite outbound filtering provider that scans the messages and will deliver them using their infrastructure/IP addresses.

Look for is to review the configurations of the HA Proxy load balancers and the Postfix mail servers to ensure they are properly configured to industry standards, and that the configuration is optimized for high load and secure.

Try and append certain header values to the message for consumption by the outbound filtering provider we are utilizing. Headers should include the sending client's IP address, sending clients host name, our MTA's hostname, the message ID, and a time stamp of when the message was received by our MTA.

Here we had fun to build a mail platform like Yahoo. You could append to the architecture many Postfix mail servers as you need.

As information Google’s Gmail use a forked version of Qmail another secure mail server.