**🧪 Lab Guide: Kubernetes Monitoring with kube-prometheus-stack & Microsoft Teams Alerts**

**🔧 Prerequisites**

* Kubernetes cluster (local or cloud)
* kubectl and helm installed
* Internet access from cluster nodes
* NodePort access enabled (or load balancer for cloud)

**📌 Step 1: Setup Helm & Namespace**

bash

CopyEdit

# Create a namespace for monitoring

kubectl create namespace monitoring

# Install Helm (if not already installed)

snap install helm --classic

# Add Prometheus community Helm chart repository

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

# Update Helm repo cache

helm repo update

**📦 Step 2: Install kube-prometheus-stack**

bash

CopyEdit

# Deploy the monitoring stack in the monitoring namespace

helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring

**📡 Step 3: Verify Deployment**

bash

CopyEdit

kubectl get all -n monitoring

kubectl get svc -n monitoring

**🌐 Step 4: Expose Grafana via NodePort**

bash

CopyEdit

kubectl edit svc prometheus-grafana -n monitoring

Change:

yaml

CopyEdit

type: ClusterIP

To:

yaml

CopyEdit

type: NodePort

Save and exit, or force replace with a modified YAML file if edited offline:

bash

CopyEdit

kubectl replace --force -f /tmp/kubectl-edit-85cjo.yaml

bash

CopyEdit

kubectl get svc -n monitoring

Note the **NodePort** and access Grafana via:  
http://<NodeIP>:<NodePort>

**🔐 Step 5: Get Grafana Credentials**

bash

CopyEdit

kubectl get secret --namespace monitoring prometheus-grafana -o jsonpath="{.data.admin-user}" | base64 --decode

echo

kubectl get secret --namespace monitoring prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 --decode

Usually defaults to:

* Username: admin
* Password: prom-operator

**📊 Step 6: Import Grafana Dashboards**

Go to Grafana ➜ Dashboards ➜ **Import**

* Paste dashboard ID: 3662 (Kubernetes Cluster Monitoring)
* Also add: 15759 (K8s Node View – if not already added)

More dashboards:

* <https://github.com/dotdc/grafana-dashboards-kubernetes>
* <https://0xdc.me/blog/a-set-of-modern-grafana-dashboards-for-kubernetes/>

**🛠️ Step 7: Expose Prometheus UI (Optional)**

bash

CopyEdit

kubectl edit svc prometheus-operated -n monitoring

Change:

yaml

CopyEdit

type: ClusterIP

clusterIP: None

To:

yaml

CopyEdit

type: NodePort

# remove the line: clusterIP: None

Save or run:

bash

CopyEdit

kubectl replace --force -f /tmp/kubectl-edit-962667183.yaml

Then access Prometheus at:  
http://<NodeIP>:<Prometheus-NodePort>

**🚨 Step 8: Create Grafana Alert**

1. Open Dashboard → **Kubernetes / Views / Nodes**
2. Identify a panel for CPU usage (e.g., CPU % (w2)).
3. Click More (⋯) → **Edit** → **Alert** → Create New Alert Rule.
4. Set evaluation time range: 5m
5. Query to use:

promql

CopyEdit

100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])))

1. Remove section B and reference C = A.
2. Preview to verify graph.
3. Set pending period to: 1m
4. Save and name your alert.

**📬 Step 9: Setup Microsoft Teams Notification**

**9.1 Create Webhook in MS Teams**

1. Go to your **test\_grafana** Team.
2. Click **More Apps** ➜ Search for Incoming Webhook.
3. Add it to a specific **channel**.
4. Name it (e.g., "Grafana Alerts") and **copy** the generated webhook URL.

**9.2 Configure Grafana Notification Channel**

1. Go to Grafana ➜ **Alerting** ➜ **Contact Points**
2. Click **New Contact Point**
3. Select Webhook:
   * Name: teams-webhook
   * URL: Paste the webhook copied from Teams
   * Optional: Test message delivery
4. Save

**🧩 Step 10: Map Alert Rules to Teams**

**10.1 Add Notification Policy**

1. Go to Grafana ➜ **Alerting** ➜ **Notification Policies**
2. Click **New Policy**
3. Set:
   * Label matchers: teams=kube (or any custom label)
   * Contact point: teams-webhook
4. Save Policy

**10.2 Map Alert to Policy**

1. In your alert rule (Step 8), under **"4. Configure labels and notifications"**:
2. Add label:

yaml

CopyEdit

teams: kube

1. Preview and Save

**✅ Final Checks**

* **Grafana** accessible on browser
* **Dashboards** visible and healthy
* **Prometheus** accessible if needed
* **Alerts** firing correctly
* **Microsoft Teams** receives notification

**🔁 Optional: Cleanup**

bash

CopyEdit

helm uninstall prometheus -n monitoring

kubectl delete namespace monitoring

**🧪 Lab Guide: EFK Stack Setup on Kubernetes using Helm**

**🎯 Objective**

Set up centralized logging using:

* **Elasticsearch**: Log storage and indexing
* **Fluentd**: Log collector and shipper
* **Kibana**: Web UI for log visualization

**🔧 Prerequisites**

* A running Kubernetes cluster (local or cloud)
* kubectl and helm installed
* Cluster nodes produce container logs under /var/log/containers (default behavior for container runtimes)

**📦 Step 1: Create Namespace**

bash

CopyEdit

kubectl create namespace efk-monitoring

**📦 Step 2: Add Elastic Helm Chart Repo**

bash

CopyEdit

helm repo add elastic https://helm.elastic.co

helm repo update

**🔍 Step 3: Install Elasticsearch**

bash

CopyEdit

helm install elasticsearch elastic/elasticsearch \

--version 7.17.3 \

-n efk-monitoring \

--set persistence.enabled=false \

--set replicas=1

Notes:

* Disabling persistence is fine for non-prod or testing environments.
* replicas=1 ensures single-node cluster for simplicity.

Check pod and service status:

bash

CopyEdit

kubectl get pods -n efk-monitoring

kubectl get svc -n efk-monitoring

**📊 Step 4: Install Kibana**

bash

CopyEdit

helm install kibana elastic/kibana \

--version 7.17.3 \

-n efk-monitoring

Verify deployment:

bash

CopyEdit

kubectl get pods -n efk-monitoring

kubectl get svc -n efk-monitoring

**🌐 Step 5: Expose Kibana (NodePort)**

Edit the Kibana service to expose via NodePort:

bash

CopyEdit

kubectl edit svc -n efk-monitoring kibana-kibana

Change:

yaml

CopyEdit

type: ClusterIP

To:

yaml

CopyEdit

type: NodePort

Save and run:

bash

CopyEdit

kubectl get svc -n efk-monitoring

Note the NodePort assigned. Access Kibana in browser:

php-template

CopyEdit

http://<NodeIP>:<NodePort>

**🔁 Step 6: Deploy Fluentd**

Get Fluentd config files:

bash

CopyEdit

git clone https://github.com/ramannkhanna2/k8s-logging-efk.git

cd k8s-logging-efk

Apply the Fluentd **ConfigMap**:

bash

CopyEdit

kubectl apply -f fluentd-config-map.yaml

Apply **RBAC permissions** for Fluentd:

bash

CopyEdit

kubectl apply -f fluentd-rbac.yaml

⚠️ Make sure Fluentd runs as a DaemonSet (configured inside the YAML) and mounts log directories like /var/log/containers.

Check Fluentd pods (usually deployed in kube-system):

bash

CopyEdit

kubectl get pods -n kube-system -w

Wait until all Fluentd pods are Running.

**🔎 Step 7: Access Kibana & View Logs**

1. Open Kibana in browser.
2. Go to **Stack Management → Index Patterns**.
3. Click **Create index pattern**.
4. Enter:

markdown

CopyEdit

\*

and click **Next**.

1. Select a time field (e.g., @timestamp) or choose "I don't want to use the time filter."
2. Click **Create index pattern**.

**📈 Step 8: Explore Logs in Discover**

1. Navigate to **Discover** tab in Kibana.
2. Logs shipped by Fluentd from all namespaces/pods will appear here.
3. You can use filters to:
   * Narrow down logs per namespace, pod, container.
   * Search errors using log: error or regex.

**✅ Final Check**

Ensure:

* **Elasticsearch** is healthy (green cluster status).
* **Fluentd** is shipping logs (kubectl logs on Fluentd DaemonSet pods).
* **Kibana** is accessible and populated with logs.
* Logs are searchable and filterable.

**🧹 Optional Cleanup**

bash

CopyEdit

helm uninstall elasticsearch -n efk-monitoring

helm uninstall kibana -n efk-monitoring

kubectl delete namespace efk-monitoring

kubectl delete -f fluentd-config-map.yaml

kubectl delete -f fluentd-rbac.yaml

**📁 Repo Reference**

This lab uses configurations from:  
🔗 <https://github.com/ramannkhanna2/k8s-logging-efk.git>

**🧪 Lab Guide: ETCD Backup & Restoration in Kubernetes (Kubeadm)**

**🎯 Objective**

* Take a **snapshot of etcd** (which stores Kubernetes cluster state)
* Delete resources to simulate a failure
* **Restore etcd from snapshot** and recover the cluster

**🔧 Prerequisites**

* Kubeadm-based Kubernetes cluster
* Root access to the control plane node
* etcd running as a **static pod**
* Backup/Restore to be done on the control plane node

**📁 Step 1: Create Test Workload**

Create a few test pods to verify they persist through backup and restoration:

bash

CopyEdit

kubectl create namespace demo

kubectl run nginx1 --image=nginx -n demo

kubectl run nginx2 --image=nginx -n demo

kubectl get pods -n demo

**🔍 Step 2: Install etcdctl**

Install the etcd-client:

bash

CopyEdit

apt update && apt install -y etcd-client

Export API version:

bash

CopyEdit

export ETCDCTL\_API=3

**🔐 Step 3: Locate ETCD Certificates & Config**

You need the certificates used by etcd:

bash

CopyEdit

ls /etc/kubernetes/pki/etcd/

You should find:

* ca.crt
* server.crt
* server.key

Also confirm etcd static pod manifest and data directory:

bash

CopyEdit

ps -ef | grep etcd

Look for the --data-dir flag in output. It’s usually:

bash

CopyEdit

/var/lib/etcd

Also inspect kubelet config if needed:

bash

CopyEdit

cat /var/lib/kubelet/config.yaml

**💾 Step 4: Take ETCD Snapshot Backup**

Run the following:

bash

CopyEdit

etcdctl --endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key \

snapshot save /root/myclust.db

Verify backup:

bash

CopyEdit

ls -lh /root/myclust.db

**💣 Step 5: Simulate Disaster (Delete Pods)**

Delete all test workloads:

bash

CopyEdit

kubectl delete ns demo

Check that all pods in demo namespace are gone:

bash

CopyEdit

kubectl get pods -n demo

**♻️ Step 6: Restore the ETCD Snapshot**

⚠️ Do **NOT** manually create the directory for restoration. The snapshot restore command will do it automatically.

bash

CopyEdit

etcdctl snapshot restore /root/myclust.db \

--data-dir /var/lib/etcd-new

The command creates /var/lib/etcd-new with restored data.

**🛠️ Step 7: Update etcd Static Pod Manifest**

Modify the static pod manifest for etcd:

bash

CopyEdit

vi /etc/kubernetes/manifests/etcd.yaml

Locate the hostPath mount for etcd data:

Change from:

yaml

CopyEdit

- hostPath:

path: /var/lib/etcd

To:

yaml

CopyEdit

- hostPath:

path: /var/lib/etcd-new

Save the file. Kubelet will automatically restart etcd using the new data directory.

**⏱️ Step 8: Wait and Verify Recovery**

Wait for a few seconds to a minute while etcd restarts.

Then verify the resources are back:

bash

CopyEdit

kubectl get pods -n demo

Your nginx1 and nginx2 pods should reappear, confirming successful restoration.

**✅ Final Verification**

Check etcd logs if needed:

bash

CopyEdit

docker ps | grep etcd # or crictl ps | grep etcd

docker logs <etcd-container-id>

Check cluster status:

bash

CopyEdit

kubectl get nodes

kubectl get all -A

**🧹 Optional Cleanup**

If you're done testing:

bash

CopyEdit

kubectl delete ns demo

rm /root/myclust.db

Revert etcd manifest back to original if needed.

**🔁 Extra: Automated Script (Optional)**

Here’s a script skeleton you could extend for automation:

bash

CopyEdit

#!/bin/bash

export ETCDCTL\_API=3

etcdctl --endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key \

snapshot save /root/etcd-backup-$(date +%F-%H%M).db

**🧪 Lab Guide: Kubernetes Cluster Upgrade (v1.30.x) Using Kubeadm**

**🎯 Objective**

* Upgrade a Kubernetes cluster from an older version (e.g., v1.29.x) to **v1.30.x**
* Upgrade all control plane and worker node components (kubeadm, kubelet, kubectl)
* Ensure zero/minimal downtime with a safe upgrade plan

**📋 Prerequisites**

* Existing Kubernetes cluster (set up via kubeadm)
* Root (sudo) access on all nodes
* Backup (especially etcd) taken before proceeding
* You are **not using a managed Kubernetes service** (like EKS, GKE, AKS)

**⛑️ Step 0: Pre-Upgrade Safety**

💡 **Take etcd backup** from the control plane before upgrading.

bash

CopyEdit

etcdctl snapshot save /root/backup-before-upgrade.db \

--endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key

**📍 Step 1: Check Current Cluster Version**

bash

CopyEdit

kubectl get nodes

kubectl version --short

This helps verify the current version before upgrade.

**📦 Step 2: View Available Versions**

**On Control Plane Node**

bash

CopyEdit

sudo apt update

sudo apt-cache madison kubeadm

Look for the latest **1.30.x-1.1** version in the output (latest stable patch version of v1.30).

**🌐 Step 3: Add Kubernetes v1.30 APT Repository**

Create APT repository list for Kubernetes:

bash

CopyEdit

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | \

gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF

deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /

EOF

sudo apt update

**🎛️ Step 4: Upgrade kubeadm on Control Plane**

bash

CopyEdit

sudo apt-mark unhold kubeadm

sudo apt-get install -y kubeadm=1.30.0-1.1

sudo apt-mark hold kubeadm

Replace 1.30.0-1.1 with the **actual version** shown via apt-cache madison.

**📋 Step 5: Review Upgrade Plan**

Check what will be upgraded and validate node health:

bash

CopyEdit

sudo kubeadm upgrade plan

This shows:

* Current and target versions
* Component changes
* Preflight check results

**🛠️ Step 6: Apply Upgrade (Control Plane)**

bash

CopyEdit

sudo kubeadm upgrade apply v1.30.x

This will:

* Upgrade kube-apiserver, kube-controller-manager, kube-scheduler, and etcd (if needed)
* Update /etc/kubernetes/manifests/\*
* Back up old configs

Once successful, move to kubelet and kubectl upgrade.

**🚀 Step 7: Upgrade kubelet and kubectl (Control Plane)**

bash

CopyEdit

sudo apt-mark unhold kubelet kubectl

sudo apt-get install -y kubelet=1.30.0-1.1 kubectl=1.30.0-1.1

sudo apt-mark hold kubelet kubectl

Reload system services and restart kubelet:

bash

CopyEdit

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl restart kubelet

**🧪 Step 8: Verify Control Plane Upgrade**

bash

CopyEdit

kubectl get nodes

kubectl version --short

Ensure the control plane node now shows v1.30.x.

**🧑‍🤝‍🧑 Step 9: Upgrade Worker Nodes**

Repeat the following **on each worker node**.

**🔄 Step 9.1: Upgrade kubeadm**

bash

CopyEdit

sudo apt-mark unhold kubeadm

sudo apt-get install -y kubeadm=1.30.0-1.1

sudo apt-mark hold kubeadm

**⚙️ Step 9.2: Run kubeadm upgrade node**

bash

CopyEdit

sudo kubeadm upgrade node

This upgrades the local node configuration (i.e., kubelet configuration files).

**🔧 Step 9.3: Upgrade kubelet and kubectl**

bash

CopyEdit

sudo apt-mark unhold kubelet kubectl

sudo apt-get install -y kubelet=1.30.0-1.1 kubectl=1.30.0-1.1

sudo apt-mark hold kubelet kubectl

Then reload and restart services:

bash

CopyEdit

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl restart kubelet

**✅ Step 10: Final Validation**

Run this on any node:

bash

CopyEdit

kubectl get nodes

kubectl version --short

Expected output:

* All nodes running version v1.30.x
* All nodes in Ready state

Also check cluster functionality:

bash

CopyEdit

kubectl get pods -A

kubectl get cs # If not deprecated in your version

**🧹 Optional: Clean Up and Maintenance**

* Prune old images if needed:

bash

CopyEdit

docker system prune -a # if using Docker

crictl rmi --prune # if using containerd

* Update cluster documentation with new version details.

**📌 Summary**

| **Component** | **Action** |
| --- | --- |
| kubeadm | Installed first, runs upgrade plan |
| kubelet | Updated last, restarted after upgrade |
| kubectl | Optional, for CLI consistency |
| Control Node | Upgrade with kubeadm upgrade apply |
| Worker Node | Upgrade with kubeadm upgrade node |