

**Hands-on: Deploy Prometheus and Grafana on Kubernetes with Pod CPU/Memory Monitoring Dashboard**

This comprehensive guide provides step-by-step instructions for deploying Prometheus and Grafana on an on-premises Kubernetes cluster and creating dashboards to monitor pod CPU and memory usage.[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)

**Prerequisites and Dependencies**

Before beginning the deployment, ensure your environment meets the following requirements:

**Cluster Requirements**

* **Kubernetes cluster version**: >= 1.16.0 (recommended: 1.20+)[[4]](#fn4)[[1]](#fn1)
* **kubectl**: Configured and authenticated to your cluster[[5]](#fn5)
* **Helm 3.x**: Package manager for Kubernetes (version 3.0 or higher)[[2]](#fn2)[[6]](#fn6)
* **Cluster access**: Admin-level permissions (RBAC enabled)[[7]](#fn7)

**Resource Requirements**

The monitoring stack requires adequate cluster resources:[[8]](#fn8)[[5]](#fn5)

|  |  |  |  |
| --- | --- | --- | --- |
| Component | CPU | Memory | Storage |
| Prometheus | 500m - 1 core | 512Mi - 1Gi | 8-50Gi (persistent) |
| Grafana | 250m | 750Mi | 1Gi (persistent) |
| kube-state-metrics | 100m | 256Mi | - |
| node-exporter | 100m | 128Mi | - |

**Architecture Components**

The monitoring solution consists of several integrated components:[[1]](#fn1)[[4]](#fn4)[[2]](#fn2)

* **Prometheus Operator**: Manages Prometheus deployments using Kubernetes custom resources
* **Prometheus**: Time-series database for metrics storage and querying
* **Grafana**: Visualization and dashboarding platform
* **kube-state-metrics**: Exports Kubernetes object state metrics[[9]](#fn9)[[10]](#fn10)
* **node-exporter**: Exposes hardware and kernel metrics from cluster nodes[[11]](#fn11)[[9]](#fn9)
* **AlertManager**: Handles alerts from Prometheus (included in stack)

**Step 1: Verify Prerequisites**

**1.1 Check Kubernetes Cluster Status**

# Verify cluster is running  
kubectl cluster-info  
  
# Check cluster version  
kubectl version --short  
  
# Verify nodes are ready  
kubectl get nodes  
  
# Check available storage classes  
kubectl get storageclass

Expected output: All nodes should be in "Ready" state, and at least one storage class should be available for persistent volumes.[[12]](#fn12)[[5]](#fn5)

**1.2 Install and Verify Helm**

# Check if Helm is installed  
helm version  
  
# If not installed, install Helm 3.x  
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash  
  
# Verify installation  
helm version

Expected Helm version: v3.x.x.[[6]](#fn6)[[2]](#fn2)

**1.3 Create Monitoring Namespace**

# Create dedicated namespace for monitoring components  
kubectl create namespace monitoring  
  
# Verify namespace creation  
kubectl get namespace monitoring

Using a dedicated namespace provides better organization and resource isolation.[[3]](#fn3)[[5]](#fn5)[[6]](#fn6)

**Step 2: Deploy Prometheus and Grafana using kube-prometheus-stack**

The kube-prometheus-stack Helm chart is the recommended approach as it deploys all necessary components with proper configuration.[[13]](#fn13)[[14]](#fn14)[[2]](#fn2)

**2.1 Add Prometheus Community Helm Repository**

# Add the prometheus-community Helm repository  
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
  
# Update repository to fetch latest charts  
helm repo update  
  
# Verify repository was added  
helm repo list

**2.2 Create Custom Values File**

Create a values.yaml file to customize the deployment for your on-premises environment:[[15]](#fn15)[[2]](#fn2)

# values.yaml  
# Prometheus configuration  
prometheus:  
 prometheusSpec:  
 # Storage configuration for persistent data  
 storageSpec:  
 volumeClaimTemplate:  
 spec:  
 storageClassName: "standard" # Replace with your storage class  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 50Gi  
   
 # Resource limits for Prometheus  
 resources:  
 requests:  
 cpu: 500m  
 memory: 512Mi  
 limits:  
 cpu: 1  
 memory: 1Gi  
   
 # Retention period for metrics  
 retention: 30d  
   
 # Select all ServiceMonitors in the cluster  
 serviceMonitorSelectorNilUsesHelmValues: false  
 podMonitorSelectorNilUsesHelmValues: false  
  
# Grafana configuration  
grafana:  
 enabled: true  
   
 # Admin credentials  
 adminPassword: "admin" # Change this in production!  
   
 # Persistent storage for Grafana  
 persistence:  
 enabled: true  
 storageClassName: "standard" # Replace with your storage class  
 size: 10Gi  
   
 # Resource limits for Grafana  
 resources:  
 requests:  
 cpu: 250m  
 memory: 750Mi  
 limits:  
 cpu: 500m  
 memory: 1Gi  
  
# Enable node-exporter for node metrics  
nodeExporter:  
 enabled: true  
  
# Enable kube-state-metrics for Kubernetes object metrics  
kubeStateMetrics:  
 enabled: true  
  
# AlertManager configuration (optional)  
alertmanager:  
 enabled: true  
 alertmanagerSpec:  
 storage:  
 volumeClaimTemplate:  
 spec:  
 storageClassName: "standard"  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 10Gi

**Important Notes**:

* Replace "standard" with your actual storage class name (check with kubectl get storageclass)[[16]](#fn16)[[12]](#fn12)
* Adjust storage sizes based on your cluster size and retention requirements[[15]](#fn15)
* Change the default Grafana password in production environments[[8]](#fn8)[[5]](#fn5)

**2.3 Install kube-prometheus-stack**

# Install the complete monitoring stack  
helm install kube-prometheus-stack \  
 prometheus-community/kube-prometheus-stack \  
 --namespace monitoring \  
 --values values.yaml  
  
# Monitor the deployment progress  
kubectl get pods -n monitoring -w

Wait for all pods to reach "Running" status. This may take 2-5 minutes.[[2]](#fn2)[[6]](#fn6)

**2.4 Verify Installation**

# Check all components are running  
kubectl get all -n monitoring  
  
# Verify Prometheus is running  
kubectl get prometheus -n monitoring  
  
# Verify Grafana deployment  
kubectl get deployment kube-prometheus-stack-grafana -n monitoring  
  
# Check ServiceMonitors are created  
kubectl get servicemonitor -n monitoring  
  
# Verify PersistentVolumeClaims are bound  
kubectl get pvc -n monitoring

Expected output: All pods should be in "Running" state, and PVCs should be "Bound".[[7]](#fn7)[[2]](#fn2)

**Step 3: Access Prometheus**

**3.1 Port-Forward to Prometheus**

# Port-forward Prometheus service to local machine  
kubectl port-forward -n monitoring \  
 svc/kube-prometheus-stack-prometheus 9090:9090

Keep this terminal open. Access Prometheus at: [**http://localhost:9090**](http://localhost:9090).[[17]](#fn17)[[18]](#fn18)[[19]](#fn19)

**3.2 Verify Prometheus Targets**

In the Prometheus web UI:

1. Navigate to **Status → Targets**
2. Verify all targets are in "UP" state
3. Common targets should include:
   * kubernetes-apiservers
   * kubernetes-nodes
   * kubernetes-pods
   * kubernetes-service-endpoints
   * kube-state-metrics
   * node-exporter

**3.3 Test Basic Queries**

In the Prometheus **Graph** tab, test these queries:[[20]](#fn20)[[21]](#fn21)

# Count all pods in the cluster  
count(kube\_pod\_info)  
  
# List all nodes  
up{job="kubernetes-nodes"}  
  
# Check kube-state-metrics is working  
up{job="kube-state-metrics"}

**Step 4: Access Grafana**

**4.1 Get Grafana Admin Password**

# Retrieve Grafana admin password (if not set in values.yaml)  
kubectl get secret -n monitoring \  
 kube-prometheus-stack-grafana \  
 -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

**4.2 Port-Forward to Grafana**

Open a new terminal:

# Port-forward Grafana service to local machine  
kubectl port-forward -n monitoring \  
 svc/kube-prometheus-stack-grafana 3000:80

Access Grafana at: [**http://localhost:3000**](http://localhost:3000).[[18]](#fn18)[[22]](#fn22)[[17]](#fn17)

**4.3 Login to Grafana**

* **Username**: admin
* **Password**: (retrieved in Step 4.1 or set in values.yaml)

**4.4 Verify Prometheus Data Source**

1. Navigate to **Configuration → Data Sources** (gear icon in sidebar)
2. You should see a pre-configured Prometheus data source
3. Click on it and scroll down to click **Save & Test**
4. Expected message: "Data source is working"[[23]](#fn23)[[8]](#fn8)

**Step 5: Create Custom Dashboard for Pod CPU/Memory Monitoring**

**5.1 Create New Dashboard**

1. Click the **+** icon in the sidebar → **Dashboard**
2. Click **Add visualization**
3. Select **Prometheus** as the data source[[22]](#fn22)[[3]](#fn3)

**5.2 Add Pod CPU Usage Panel**

**Panel 1: Pod CPU Usage (Cores)**

**Configuration:**

1. In the query editor, select **Code** mode
2. Enter the following PromQL query:[[24]](#fn24)[[21]](#fn21)[[20]](#fn20)

sum(rate(container\_cpu\_usage\_seconds\_total{container!="", container!="POD"}[5m])) by (pod, namespace)

1. **Panel Settings** (right sidebar):
   * **Title**: "Pod CPU Usage (Cores)"
   * **Panel type**: Time series
   * **Legend**: {{namespace}}/{{pod}}
2. **Standard Options**:
   * **Unit**: "cores"
   * **Decimals**: 3
3. Click **Apply**

**Query Explanation**:[[25]](#fn25)[[21]](#fn21)[[20]](#fn20)

* container\_cpu\_usage\_seconds\_total: Cumulative CPU time consumed
* rate(...[5m]): Per-second rate over 5 minutes
* container!="": Exclude pause containers
* sum(...) by (pod, namespace): Aggregate by pod and namespace

**Panel 2: Pod CPU Usage (Percentage of Limits)**

1. Click **Add → Visualization**
2. Enter this query:[[26]](#fn26)[[24]](#fn24)[[20]](#fn20)

100 \* (  
 sum(rate(container\_cpu\_usage\_seconds\_total{container!="", container!="POD"}[5m])) by (pod, namespace)  
 /  
 sum(kube\_pod\_container\_resource\_limits{resource="cpu"}) by (pod, namespace)  
)

1. **Panel Settings**:
   * **Title**: "Pod CPU Usage (% of Limits)"
   * **Unit**: "percent (0-100)"
   * **Legend**: {{namespace}}/{{pod}}
2. **Thresholds** (optional):
   * Green: 0-70
   * Yellow: 70-90
   * Red: 90-100
3. Click **Apply**

**5.3 Add Pod Memory Usage Panels**

**Panel 3: Pod Memory Usage (Bytes)**

1. Click **Add → Visualization**
2. Enter this query:[[21]](#fn21)[[24]](#fn24)[[20]](#fn20)

sum(container\_memory\_working\_set\_bytes{container!="", container!="POD"}) by (pod, namespace)

1. **Panel Settings**:
   * **Title**: "Pod Memory Usage"
   * **Unit**: "bytes (SI)"
   * **Legend**: {{namespace}}/{{pod}}
2. Click **Apply**

**Query Explanation**:[[20]](#fn20)[[21]](#fn21)

* container\_memory\_working\_set\_bytes: Current active memory usage
* This is the metric used by Kubernetes for OOM decisions

**Panel 4: Pod Memory Usage (Percentage of Limits)**

1. Click **Add → Visualization**
2. Enter this query:[[24]](#fn24)[[21]](#fn21)

100 \* (  
 sum(container\_memory\_working\_set\_bytes{container!="", container!="POD"}) by (pod, namespace)  
 /  
 sum(kube\_pod\_container\_resource\_limits{resource="memory"}) by (pod, namespace)  
)

1. **Panel Settings**:
   * **Title**: "Pod Memory Usage (% of Limits)"
   * **Unit**: "percent (0-100)"
   * **Legend**: {{namespace}}/{{pod}}
2. **Thresholds**:
   * Green: 0-70
   * Yellow: 70-85
   * Red: 85-100
3. Click **Apply**

**5.4 Add Dashboard Variables (Optional but Recommended)**

Variables allow dynamic filtering by namespace and pod.[[22]](#fn22)

1. Click **Dashboard settings** (gear icon at top)
2. Select **Variables** → **Add variable**

**Variable 1: Namespace**

* **Name**: namespace
* **Type**: Query
* **Data source**: Prometheus
* **Query**: label\_values(kube\_pod\_info, namespace)
* **Multi-value**: Enable
* **Include All option**: Enable
* Click **Apply**

**Variable 2: Pod**

* **Name**: pod
* **Type**: Query
* **Data source**: Prometheus
* **Query**: label\_values(kube\_pod\_info{namespace=~"$namespace"}, pod)
* **Multi-value**: Enable
* **Include All option**: Enable
* Click **Apply**

1. Return to dashboard and modify queries to use variables:

# CPU query with variables  
sum(rate(container\_cpu\_usage\_seconds\_total{  
 namespace=~"$namespace",   
 pod=~"$pod",  
 container!="",   
 container!="POD"  
}[5m])) by (pod, namespace)  
  
# Memory query with variables  
sum(container\_memory\_working\_set\_bytes{  
 namespace=~"$namespace",   
 pod=~"$pod",  
 container!="",   
 container!="POD"  
}) by (pod, namespace)

**5.5 Add Additional Useful Panels**

**Panel 5: Top 10 CPU Consuming Pods**

topk(10, sum(rate(container\_cpu\_usage\_seconds\_total{container!="", container!="POD"}[5m])) by (pod, namespace))

**Panel type**: Bar chart[[20]](#fn20)

**Panel 6: Top 10 Memory Consuming Pods**

topk(10, sum(container\_memory\_working\_set\_bytes{container!="", container!="POD"}) by (pod, namespace))

**Panel type**: Bar chart

**Panel 7: Pod Restart Count**

sum(kube\_pod\_container\_status\_restarts\_total) by (namespace, pod)

**Panel type**: Stat[[27]](#fn27)

**Panel 8: Pods Not Ready**

sum(kube\_pod\_status\_phase{phase!="Running"}) by (namespace, pod, phase)

**Panel type**: Table

**5.6 Save Dashboard**

1. Click **Save dashboard** icon (floppy disk) at top
2. Name: "Kubernetes Pod Monitoring"
3. Click **Save**

**Step 6: Import Pre-built Dashboards (Optional)**

Grafana community provides many pre-built Kubernetes dashboards.[[28]](#fn28)[[23]](#fn23)

**6.1 Import Dashboard by ID**

1. In Grafana, click **+ → Import**
2. Enter one of these popular dashboard IDs:[[29]](#fn29)[[30]](#fn30)[[31]](#fn31)[[32]](#fn32)
   * **17149**: Kubernetes Pod Metrics
   * **15760**: Kubernetes Views / Pods
   * **9144**: Kubernetes Pod Monitoring
   * **6417**: Kubernetes Cluster Monitoring
3. Click **Load**
4. Select your Prometheus data source
5. Click **Import**

**6.2 Import Dashboard from JSON**

If you have a dashboard JSON file:

1. Click **+ → Import**
2. Click **Upload JSON file** or paste JSON
3. Select Prometheus data source
4. Click **Import**[[33]](#fn33)[[23]](#fn23)[[28]](#fn28)

**Step 7: Configure ServiceMonitor for Custom Applications**

To monitor your own applications, create ServiceMonitor resources.[[34]](#fn34)[[35]](#fn35)[[36]](#fn36)

**7.1 Example: Deploy Sample Application**

# sample-app-deployment.yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: example-app  
 namespace: default  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: example-app  
 template:  
 metadata:  
 labels:  
 app: example-app  
 spec:  
 containers:  
 - name: example-app  
 image: quay.io/brancz/prometheus-example-app:v0.5.0  
 ports:  
 - name: metrics  
 containerPort: 8080  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: example-app  
 namespace: default  
 labels:  
 app: example-app  
spec:  
 selector:  
 app: example-app  
 ports:  
 - name: metrics  
 port: 8080  
 targetPort: 8080

kubectl apply -f sample-app-deployment.yaml

**7.2 Create ServiceMonitor**

# sample-app-servicemonitor.yaml  
apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: example-app  
 namespace: monitoring # Must be in same namespace as Prometheus  
 labels:  
 release: kube-prometheus-stack # Must match Prometheus selector  
spec:  
 selector:  
 matchLabels:  
 app: example-app  
 namespaceSelector:  
 matchNames:  
 - default  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics

kubectl apply -f sample-app-servicemonitor.yaml

**7.3 Verify ServiceMonitor**

# Check ServiceMonitor was created  
kubectl get servicemonitor -n monitoring  
  
# Verify in Prometheus UI: Status → Targets  
# Look for "serviceMonitor/monitoring/example-app/0"

**Step 8: Best Practices and Optimization**

**8.1 Resource Optimization**

**Adjust retention based on needs**:[[15]](#fn15)

prometheus:  
 prometheusSpec:  
 retention: 15d # Reduce for less storage  
 retentionSize: 45GB # Or limit by size

**Configure resource limits properly**:[[15]](#fn15)

prometheus:  
 prometheusSpec:  
 resources:  
 requests:  
 cpu: 1  
 memory: 2Gi  
 limits:  
 cpu: 2  
 memory: 4Gi

**8.2 High Availability Configuration**

For production deployments, enable HA:[[15]](#fn15)

prometheus:  
 prometheusSpec:  
 replicas: 2  
 podAntiAffinity: hard  
  
alertmanager:  
 alertmanagerSpec:  
 replicas: 3

**8.3 Persistent Storage Best Practices**

**Use storage classes with retain policy**:[[37]](#fn37)

# Check reclaim policy  
kubectl get pv  
  
# For production, ensure ReclaimPolicy is "Retain"

**Regular backups**:

# Backup Prometheus data  
kubectl exec -n monitoring prometheus-kube-prometheus-stack-prometheus-0 -- \  
 tar czf /tmp/prometheus-backup.tar.gz /prometheus  
  
# Copy to local  
kubectl cp monitoring/prometheus-kube-prometheus-stack-prometheus-0:/tmp/prometheus-backup.tar.gz \  
 ./prometheus-backup.tar.gz

**8.4 Query Optimization**

**Use appropriate time ranges**:[[20]](#fn20)

# For real-time: 1-5 minutes  
rate(container\_cpu\_usage\_seconds\_total[1m])  
  
# For trends: 10-15 minutes  
rate(container\_cpu\_usage\_seconds\_total[15m])

**Filter early in queries**:[[20]](#fn20)

# Good - filter first  
sum(rate(container\_cpu\_usage\_seconds\_total{namespace="production"}[5m]))  
  
# Less efficient - filter after aggregation  
sum(rate(container\_cpu\_usage\_seconds\_total[5m])) by (namespace) {namespace="production"}

**8.5 Dashboard Export and Version Control**

**Export dashboard as JSON**:

1. Open dashboard
2. Click **Dashboard settings** (gear icon)
3. Select **JSON Model**
4. Copy JSON and save to file

**Import in new environment**:

# Use Grafana API to import  
curl -X POST http://localhost:3000/api/dashboards/db \  
 -H "Content-Type: application/json" \  
 -d @dashboard.json

**Troubleshooting Common Issues**

**Issue 1: Pods Not Starting**

# Check pod status  
kubectl get pods -n monitoring  
  
# View pod logs  
kubectl logs -n monitoring <pod-name>  
  
# Describe pod for events  
kubectl describe pod -n monitoring <pod-name>

**Common causes**:

* Insufficient resources: Increase node capacity or reduce resource requests[[8]](#fn8)
* Storage class issues: Verify storage class exists and is default[[12]](#fn12)[[16]](#fn16)
* Image pull errors: Check image availability and credentials

**Issue 2: Prometheus Not Scraping Targets**

# Check ServiceMonitor configuration  
kubectl get servicemonitor -n monitoring <servicemonitor-name> -o yaml  
  
# Verify Prometheus configuration  
kubectl get prometheus -n monitoring -o yaml  
  
# Check Prometheus logs  
kubectl logs -n monitoring prometheus-kube-prometheus-stack-prometheus-0

**Common causes**:[[38]](#fn38)[[34]](#fn34)

* Label mismatch: Ensure ServiceMonitor labels match Prometheus selector
* Wrong namespace: ServiceMonitor must be in correct namespace
* Network policies: Check if network policies block traffic

**Issue 3: Grafana Data Source Not Working**

# Test Prometheus connectivity from Grafana pod  
kubectl exec -n monitoring deployment/kube-prometheus-stack-grafana -- \  
 curl http://kube-prometheus-stack-prometheus:9090/api/v1/query?query=up  
  
# Check Grafana logs  
kubectl logs -n monitoring deployment/kube-prometheus-stack-grafana

**Common causes**:[[39]](#fn39)[[33]](#fn33)

* Wrong URL: Verify Prometheus service name and port
* DNS issues: Check CoreDNS is working
* Data source configuration: Recreate data source with correct settings

**Issue 4: No Metrics Showing**

**Verify metrics are being collected**:

# Check if metrics endpoints are exposed  
kubectl get endpoints -n monitoring  
  
# Test metrics endpoint directly  
kubectl port-forward -n monitoring svc/kube-prometheus-stack-kube-state-metrics 8080:8080  
curl http://localhost:8080/metrics

**Verify queries in Prometheus**:[[21]](#fn21)[[20]](#fn20)

# Test basic query  
up  
  
# Check specific metrics exist  
container\_cpu\_usage\_seconds\_total  
container\_memory\_working\_set\_bytes

**Issue 5: Storage Issues**

# Check PVC status  
kubectl get pvc -n monitoring  
  
# Check PV status  
kubectl get pv  
  
# Describe PVC for events  
kubectl describe pvc -n monitoring <pvc-name>

**Solutions**:

* Create appropriate storage class if missing
* Increase storage size in values.yaml and upgrade
* Check storage provisioner is running

**Additional Resources and Next Steps**

**Recommended Pre-built Dashboards**

* **6417**: Kubernetes Cluster Monitoring (Global Stats)
* **15760**: Kubernetes Views / Pods
* **17149**: Kubernetes Pod Metrics
* **15661**: Kubernetes Views / Nodes
* **13770**: Kubernetes apiserver[[30]](#fn30)[[31]](#fn31)[[29]](#fn29)

**Advanced Topics**

1. **Configure Alerting Rules**: Create PrometheusRule resources for automated alerts
2. **Integrate with Slack/Email**: Configure AlertManager for notifications
3. **Multi-cluster Monitoring**: Use Thanos or Cortex for federation
4. **Custom Exporters**: Deploy application-specific metric exporters
5. **Recording Rules**: Pre-compute expensive queries for better performance

**Documentation References**

* Prometheus Operator: <https://prometheus-operator.dev/>[[40]](#fn40)[[1]](#fn1)
* Grafana Documentation: <https://grafana.com/docs/>[[8]](#fn8)
* PromQL Guide: <https://prometheus.io/docs/prometheus/latest/querying/basics/>
* kube-prometheus-stack Chart: <https://github.com/prometheus-community/helm-charts>[[14]](#fn14)

**Learning Resources**

* **Prometheus Query Examples**: Practice PromQL with real cluster data[[27]](#fn27)[[20]](#fn20)
* **Grafana Tutorials**: Learn advanced dashboard features
* **Kubernetes Metrics**: Understand kube-state-metrics vs. cAdvisor[[10]](#fn10)[[41]](#fn41)

**Conclusion**

You now have a complete Prometheus and Grafana monitoring stack deployed on your on-premises Kubernetes cluster with custom dashboards for tracking pod CPU and memory usage. This setup provides:

* **Real-time visibility**: Monitor resource consumption across all pods
* **Historical analysis**: Persistent storage allows trend analysis[[15]](#fn15)
* **Customizable dashboards**: Create views tailored to your needs[[3]](#fn3)[[22]](#fn22)
* **Extensible monitoring**: Add new ServiceMonitors for applications[[34]](#fn34)[[38]](#fn38)
* **Production-ready**: Configured with persistent storage and proper resource limits

The monitoring solution can be extended with alerting rules, custom exporters, and integration with incident management systems as your needs grow.[[4]](#fn4)[[1]](#fn1)[[2]](#fn2)

⁂

1. <https://prometheus-operator.dev/docs/getting-started/installation/>

1. <https://spacelift.io/blog/prometheus-operator>

1. <https://devopscube.com/setup-grafana-kubernetes/>

1. <https://github.com/prometheus-operator/prometheus-operator>

1. <https://grafana.com/docs/grafana/latest/setup-grafana/installation/kubernetes/>

1. <https://grafana.com/docs/grafana/latest/setup-grafana/installation/helm/>

1. <https://grafana.com/docs/grafana-cloud/monitor-infrastructure/kubernetes-monitoring/configuration/config-other-methods/prometheus/prometheus-operator/>

1. <https://www.plural.sh/blog/grafana-kubernetes-monitoring/>

1. <https://www.ibm.com/docs/en/gdsc/3.x?topic=configuring-prometheus-node-exporter-kubernetes>

1. <https://spacelift.io/blog/kube-state-metrics>

1. <https://www.sysdig.com/blog/kubernetes-metrics-ingestion>

1. <https://github.com/prometheus-community/helm-charts/issues/186>

1. <https://spacelift.io/blog/prometheus-kubernetes>

1. <https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack>

1. <https://devopscube.com/setup-prometheus-helm-chart/>

1. <https://stackoverflow.com/questions/56391693/how-to-enable-persistence-in-helm-prometheus-operator>

1. <https://stackoverflow.com/questions/67373856/unable-to-access-prometheus-dashboard-port-forwarding-doesnt-work>

1. <https://www.civo.com/learn/connecting-to-grafana-deployed-to-kubernetes-via-prometheus-operator-in-marketplace>

1. <https://docs.replicated.com/enterprise/monitoring-access-dashboards>

1. <https://signoz.io/guides/prometheus-queries-to-get-cpu-and-memory-usage-in-kubernetes-pods/>

1. <https://www.baeldung.com/ops/prometheus-cpu-memory-kubernetes>

1. <https://www.teamim.com/insights/pod-monitoring-with-prometheus-grafana>

1. <https://prometheus.io/docs/visualization/grafana/>

1. <https://stackoverflow.com/questions/55143656/prometheus-queries-to-get-cpu-and-memory-usage-in-kubernetes-pods>

1. <https://betterstack.com/community/questions/how-to-calculate-cpu-usage-in-kubernetes-using-prometheus/>

1. <https://stackoverflow.com/questions/40327062/how-to-calculate-containers-cpu-usage-in-kubernetes-with-prometheus-as-monitori>

1. <https://www.sysdig.com/blog/prometheus-query-examples>

1. <https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/import-dashboards/>

1. <https://grafana.com/grafana/dashboards/21382-pod-resources-stats/>

1. <https://grafana.com/grafana/dashboards/20372-kubernetes-workloads-cpu-and-memory-metrics/>

1. <https://grafana.com/grafana/dashboards/21298-kubernetes-pod-dashboard/>

1. <https://grafana.com/grafana/dashboards/17149-kubernetes-pod-metrics/>

1. <https://stackoverflow.com/questions/54776604/how-can-i-change-the-datasource-for-a-grafana-dashboard>

1. <https://prometheus-operator.dev/docs/developer/getting-started/>

1. <https://documentation.suse.com/cloudnative/rancher-manager/latest/en/observability/monitoring-and-dashboards/configuration/servicemonitors-and-podmonitors.html>

1. <https://observability.thomasriley.co.uk/prometheus/configuring-prometheus/using-service-monitors/>

1. <https://romikoderbynew.com/2020/10/19/kubernetes-prometheus-use-an-existing-persistent-volume-claim/>

1. <https://coralogix.com/guides/kubernetes-monitoring/kubernetes-service-monitoring/>

1. <https://community.grafana.com/t/automated-import-of-grafana-dashboard-not-finding-prometheus-datasource/53929>

1. <https://prometheus-operator.dev>

1. <https://github.com/kubernetes/kube-state-metrics>

1. <https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/>

1. <https://www.youtube.com/watch?v=qMKSBUkSRYc>

1. <https://docs.aws.amazon.com/eks/latest/userguide/deploy-prometheus.html>

1. <https://artifacthub.io/packages/helm/choerodon/prometheus-operator>

1. <https://www.youtube.com/watch?v=sgYrEleW24E>

1. <https://prometheus.io/docs/prometheus/latest/installation/>

1. <https://grafana.com/docs/alloy/latest/set-up/install/kubernetes/>

1. <https://www.reddit.com/r/kubernetes/comments/1i8z5xa/how_do_people_deploy_a_prometheus_stack/>

1. <https://www.metricfire.com/blog/deploying-grafana-to-kubernetes/>

1. <https://ranchermanager.docs.rancher.com/reference-guides/monitoring-v2-configuration/servicemonitors-and-podmonitors>

1. <https://www.reddit.com/r/kubernetes/comments/y5pawd/how_do_you_monitor_cpumemory_usage_of_pod_with/>

1. <https://www.virtana.com/glossary/what-is-ksm-exporter-kube-state-metrics-exporter/>

1. <https://learn.microsoft.com/en-us/azure/azure-monitor/containers/prometheus-metrics-scrape-crd>

1. <https://grafana.com/grafana/dashboards/21225-kubernetes-statefulset-cpu-and-memory-metrics/>

1. <https://github.com/prometheus/node_exporter>

1. <https://www.reddit.com/r/grafana/comments/flridv/variables_in_provisioned_dashboard_json_file/>

1. <https://community.temporal.io/t/prometheus-persistence/8717>

1. <https://community.grafana.com/t/dashboard-import-with-datasource-variable/4896>

1. <https://grafana.com/blog/2023/08/30/grafana-json-api-how-to-import-third-party-data-sources-in-grafana-cloud/>

1. <https://www.reddit.com/r/kubernetes/comments/10ytb5r/prometheus_helm_defining_storagclass_for_server/>

1. <https://earthly.dev/blog/grafana-and-prometheus-k8s/>

1. <https://stackoverflow.com/questions/62770744/promql-query-to-find-cpu-and-memory-used-for-the-last-week>

1. <https://github.com/prometheus-operator/kube-prometheus/issues/2522>

1. <https://home.robusta.dev/blog/3-common-mistakes-with-promql-and-kubernetes-metrics>

1. <https://grafana.com/grafana/dashboards/9144-kubernetes-pod-monitoring/>

1. <https://last9.io/blog/monitoring-container-cpu-usage/>

1. <https://learn.microsoft.com/en-us/azure/azure-monitor/metrics/prometheus-api-promql>

1. <https://grafana.com/grafana/dashboards/747-pod-metrics/>

1. <https://signoz.io/guides/how-to-calculate-containers-cpu-usage-in-kubernetes-with-prometheus-as-monitoring/>

1. <https://docs.logz.io/docs/user-guide/infrastructure-monitoring/introduction-to-prometheus/promql-query/>

1. <https://grafana.com/grafana/dashboards/18283-kubernetes-dashboard/>

1. <https://groups.google.com/g/prometheus-users/c/IXgOrtesGEo>

1. <https://grafana.com/grafana/dashboards/15760-kubernetes-views-pods/>

1. <https://grafana.com/grafana/dashboards/16697-kubernetes-pod-monitoring/>