Got it. Below is a practical, end-to-end playbook to make **Grafana (running in cluster3 via Helm)** query **per-namespace, per-cluster** metrics from **Thanos Querier** on each of your 10 OpenShift clusters — **without granting cluster-wide access**. It uses **namespace-scoped RBAC** and, where needed, **a service account per namespace** so a namespace’s datasource can’t see other namespaces.

Quick background (why this works):

* OpenShift’s Thanos Querier exposes two important ports:
  + **9091 (“web”)** → needs cluster-monitoring-view (cluster-wide) and shows *all* namespaces.
  + **9092 (“tenancy”)** → **project-scoped** with kube-rbac-proxy; access is limited to the namespace(s) you authorize and (in practice) you pass namespace=<ns> as a **query parameter**. This is how the OpenShift web console itself enforces multi-tenancy. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com), [Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier), [blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

**Architecture you’ll deploy**

* **Each monitored cluster (1..10):**
  1. User-workload monitoring enabled (so app metrics get scraped).
  2. A **Route** that exposes Thanos Querier’s **tenancy port (9092)**.
  3. For every namespace you want Grafana to see (e.g., team-a, team-b), create a **service account** (e.g., grafana-thanos) and bind **namespace-scoped view**.
  4. Generate a **bearer token** for each service account (used by Grafana’s data source for that namespace).
* **Cluster3 (Grafana):**
  1. One Grafana instance (your Helm release) with **one Prometheus data source per (cluster, namespace)** pointing at the target cluster’s thanos-querier **tenancy endpoint** and using the **namespace’s service-account token**.
  2. Data source sets **Custom query parameters**: namespace=<ns>. (Grafana operator CRD historically didn’t persist this setting; with Helm you can provision it directly. If you do use the operator, be aware of that quirk.) ([blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

Yes — **one Grafana in cluster3 can serve all 10 clusters**, provided you meet these requirements: **network/DNS reachability** to each cluster’s Thanos Route, **TLS trust** (use the cluster CA or a real cert), and **a dedicated, namespace-scoped token per data source**. Thanos’s query layer is designed to aggregate many backends; here we’re just talking to each cluster’s Thanos directly. ([Thanos](https://thanos.io/tip/components/query.md/?utm_source=chatgpt.com))

**Step 1 — Enable user-workload monitoring on every cluster**

Run once per cluster (1..10).

# cluster-monitoring-config.yaml

# Enables user-workload monitoring stack so your app namespaces get scraped.

apiVersion: v1

kind: ConfigMap

metadata:

name: cluster-monitoring-config

namespace: openshift-monitoring

data:

config.yaml: |

enableUserWorkload: true

---

# user-workload-config.yaml

# Placeholder ConfigMap for optional UWM tuning; can be empty.

apiVersion: v1

kind: ConfigMap

metadata:

name: user-workload-config

namespace: openshift-user-workload-monitoring

data:

config.yaml: |

{}

oc apply -f cluster-monitoring-config.yaml

oc apply -f user-workload-config.yaml

(Documentation confirms this setup for accessing metrics and tenant scoping; we’ll use the tenancy port in later steps.) ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier), [Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))

**Step 2 — Expose the tenancy endpoint (9092) via Route**

Default “thanos-querier” Route usually targets the **web (9091)** port (cluster-wide). Create an **additional Route** to target the **tenancy** port (9092) so Grafana can hit it from cluster3 over HTTPS.

# thanos-tenancy-route.yaml

# Creates a separate Route to the 'tenancy' port (9092).

# OAuth is NOT needed here; kube-rbac-proxy on 9092 enforces namespace-level RBAC.

apiVersion: route.openshift.io/v1

kind: Route

metadata:

name: thanos-querier-tenancy

namespace: openshift-monitoring

annotations:

# Reencrypt to keep TLS on both edges. Adjust as per your TLS policy.

haproxy.router.openshift.io/timeout: 5m

spec:

host: thanos-tenancy.<your-cluster-domain> # e.g., thanos-tenancy.apps.cluster1.example.com

to:

kind: Service

name: thanos-querier

weight: 100

port:

targetPort: tenancy # This selects the 9092 service port

tls:

termination: reencrypt

insecureEdgeTerminationPolicy: Redirect

oc -n openshift-monitoring apply -f thanos-querier-tenancy.yaml

oc -n openshift-monitoring get route thanos-querier-tenancy

Why 9092? Red Hat docs describe 9092 as the **project-scoped** API that honors **namespace RBAC**; the community & Red Hat blogs call out using namespace=<ns> with this port. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com), [Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier), [blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

**Step 3 — Namespace-scoped access: ServiceAccount, RoleBinding (per namespace to monitor)**

Do this **in each target namespace** you want Grafana to see (on each monitored cluster).  
You can reuse the same SA for multiple namespaces, but for *strongest isolation* use **one SA per namespace**.

# ns-reader.yaml

# Create a service account dedicated to Grafana -> Thanos queries for THIS namespace only.

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos

namespace: team-a

---

# Bind the built-in "view" ClusterRole in THIS namespace to the service account.

# This lets kube-rbac-proxy on 9092 authorize queries only for 'team-a'.

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: team-a

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: team-a

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

oc apply -f ns-reader.yaml

The **9092** endpoint authorizes requests via **kube-rbac-proxy + SubjectAccessReview**; binding view in the project is the documented minimum to query metrics for that project via 9092. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))

**Step 4 — Generate a bearer token for Grafana’s data source**

Use **short/medium-lived tokens** and rotate them periodically (automation recommended).

# One token per (cluster, namespace) SA.

# Example for cluster1 / namespace team-a:

oc -n team-a create token grafana-thanos --duration=24h > team-a.cluster1.token

Note: Kubernetes SA tokens are *expiring* by default; plan for rotation. Alternatives include putting Grafana inside the target cluster (so it can mount a projected SA token), or fronting 9092 with your own auth proxy that issues long-lived credentials. The standard, supported path for remote access is still to use **9092 + RBAC**. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))

**Step 5 — Provision Grafana (Helm) with one Prometheus data source per (cluster, namespace)**

For Helm, use a datasources.yaml in values.yaml (grafana.additionalDataSources or provisioning files).  
**Key points**:  
• URL = your **9092 Route** from the target cluster  
• **Custom query parameters** = namespace=<ns>  
• **Authorization** header = Bearer <token from Step 4>  
• **TLS** = trust the route’s cert (set tlsSkipVerify: true only if you must)

**Example: two namespaces (team-a, team-b) on cluster1, one namespace (payments) on cluster2**

# values.yaml (excerpt for Grafana Helm chart)

grafana:

grafana.ini:

server:

root\_url: %(protocol)s://%(domain)s/

additionalDataSources:

# ---------------- cluster1 / team-a ----------------

- name: ocp-cluster1-team-a

type: prometheus

url: https://thanos-tenancy.apps.cluster1.example.com # Route to 9092 on cluster1

access: proxy

isDefault: false

jsonData:

# IMPORTANT: pass the namespace selector

customQueryParameters: "namespace=team-a"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER1\_TEAM\_A\_TOKEN}" # set via Helm --set or Secret mount

# ---------------- cluster1 / team-b ----------------

- name: ocp-cluster1-team-b

type: prometheus

url: https://thanos-tenancy.apps.cluster1.example.com

access: proxy

isDefault: false

jsonData:

customQueryParameters: "namespace=team-b"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER1\_TEAM\_B\_TOKEN}"

# ---------------- cluster2 / payments ----------------

- name: ocp-cluster2-payments

type: prometheus

url: https://thanos-tenancy.apps.cluster2.example.com

access: proxy

isDefault: false

jsonData:

customQueryParameters: "namespace=payments"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER2\_PAYMENTS\_TOKEN}"

Why customQueryParameters? OpenShift’s **tenancy path** expects namespace=<ns>; that’s what the web console does under the hood. Older grafana-operator CRDs didn’t persist this field — with Helm provisioning you can reliably set it. ([blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

Deploy/update Grafana:

# Supply tokens via environment or a Secret mounted as envFrom; here is an example with --set-file:

helm upgrade --install grafana grafana/grafana \

-n grafana \

-f values.yaml \

--set-file "grafana.envValueFrom[0].name=CLUSTER1\_TEAM\_A\_TOKEN,grafana.envValueFrom[0].valueFrom.secretKeyRef.name=cluster1-team-a-token-secret,grafana.envValueFrom[0].valueFrom.secretKeyRef.key=token" \

# ...repeat for other tokens or manage via Kubernetes Secrets you template with Helm

**Step 6 — (Optional) Lock down ingress and trust**

* **Restrict who can reach the 9092 Route** (e.g., use a dedicated IngressController with an allowlist or mTLS).
* **Inject trusted CA bundle** into Grafana if your Routes use a custom CA. (You can mount the OpenShift trust bundle as a ConfigMap; the Red Hat blog shows injecting trusted CAs for OAuth/Grafana proxies.) ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier))

**Example: Full namespace-scoped pack per cluster**

Repeat this skeleton for each **namespace** you want Grafana to see across your 10 clusters.

# 01-ns-reader.yaml (run on target cluster)

# Purpose: ServiceAccount + RoleBinding(view) scoped to THIS namespace only.

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos

namespace: my-namespace

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: my-namespace

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: my-namespace

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

# 02-token.sh (generate a token; store it securely for Grafana)

oc -n my-namespace create token grafana-thanos --duration=24h

# 03-tenancy-route.yaml (run once per cluster; shared by all namespaces)

# Purpose: Exposes Thanos Querier tenancy port (9092) externally via TLS Route.

apiVersion: route.openshift.io/v1

kind: Route

metadata:

name: thanos-querier-tenancy

namespace: openshift-monitoring

spec:

host: thanos-tenancy.apps.clusterX.example.com

to:

kind: Service

name: thanos-querier

port:

targetPort: tenancy

tls:

termination: reencrypt

insecureEdgeTerminationPolicy: Redirect

# 04-grafana-datasource.yaml (cluster3 Helm provisioning snippet)

# Purpose: Define a Prometheus datasource for THIS (cluster, namespace) pair.

- name: ocp-clusterX-my-namespace

type: prometheus

url: https://thanos-tenancy.apps.clusterX.example.com

access: proxy

jsonData:

customQueryParameters: "namespace=my-namespace" # <-- Enforces namespace tenancy

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTERX\_MY\_NS\_TOKEN}"

**Using the datasources in dashboards**

Nothing special in PromQL—just write queries as usual (e.g., sum(rate(container\_cpu\_usage\_seconds\_total[5m])) by (pod)). The **tenancy layer** makes sure the selectors only return series from the namespace tied to the data source. If you accidentally query without a namespace matcher, you’ll still only get **that namespace**, thanks to 9092 + RBAC. (If you ever target 9091, you bypass that and would need cluster-wide rights — avoid for isolation.) ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))

**FAQs / Gotchas**

* **Can I avoid per-namespace service accounts and use Roles/RoleBindings instead of cluster roles?**  
  Yes — that’s what we’re doing. We’re *not* using cluster-monitoring-view. We only bind **view in a namespace** to the SA used by the datasource. The kube-rbac-proxy on 9092 enforces it. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))
* **Do I have to pass namespace=<ns>?**  
  In practice, yes — this mirrors how the OpenShift console calls the tenancy endpoint; it’s also called out in the Red Hat and community posts. Some tooling auto-adds it, but Grafana doesn’t, so set it in the datasource. ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier), [blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))
* **Token lifetime/rotation?**  
  SA tokens created via oc create token expire. Automate rotation (e.g., a small Job/Secret refresh + helm upgrade) or consider hosting Grafana in the same cluster and mounting a projected token. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))
* **Can Grafana in cluster3 handle all 10 clusters?**  
  Yes — Thanos/Grafana are built for this. Ensure **network reachability** to each cluster’s 9092 Route, **TLS trust**, and **sizing** (Grafana resources and backend concurrency) as your dashboards grow. ([Thanos](https://thanos.io/tip/components/query.md/?utm_source=chatgpt.com))

**References (key points we relied on)**

* **Why 9091 vs 9092 & RBAC differences** — Red Hat docs (“Accessing metrics”): 9091 requires cluster-monitoring-view (all projects); **9092** is project-scoped; bind **view** in that project. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/monitoring/accessing-metrics?utm_source=chatgpt.com))
* **Passing namespace= param & tenancy behavior** — Red Hat blog and detailed community walk-through. ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier), [blog.stderr.at](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))
* **Thanos as a unified query layer** — Thanos docs. ([Thanos](https://thanos.io/tip/components/query.md/?utm_source=chatgpt.com))

If you want, tell me the **cluster base domains** and the **list of (cluster, namespace)** pairs you plan to monitor, and I’ll output a **ready-to-apply bundle** (Routes + SA/RBAC + Grafana datasource provisioning) tailored for your exact 10 clusters.

Awesome—let’s wire this up cleanly and safely.

Below is a **step-by-step** plan (with fully commented YAML you can paste/apply) to let **Grafana in cluster3** query **only the target namespace metrics** from each OpenShift cluster by hitting **Thanos Querier’s tenancy port (9092)** with **namespace-scoped RBAC**. I’ll also include concrete examples for your two cases: **abc1 on cluster1** and **xyz on cluster4**.

Why this approach:

* OpenShift runs **two Thanos Querier ports** that matter:
  + **9091 “web”** → requires cluster-monitoring-view and returns **all** namespaces (not desired for isolation).
  + **9092 “tenancy”** → fronted by kube-rbac-proxy; **enforces project (namespace) RBAC** and expects **namespace=<ns>** query parameter. This is what the OpenShift console uses for per-project queries. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))
* The Red Hat blog & slides reiterate: use **9092 + namespace= + namespace-level view** for strict isolation. ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier?utm_source=chatgpt.com))

Yes—**one Grafana instance in cluster3 can serve all 10 clusters** as long as cluster3 can reach each cluster’s 9092 endpoint over the network and trusts the TLS certificates. (Thanos is designed to be queried remotely; we’re just calling each cluster’s Thanos.) ([Red Hat](https://www.redhat.com/tracks/_pfcdn/assets/10330/contents/548954/4c855e59-fd2a-49f8-8b84-76b31ccb2ef7.pdf?utm_source=chatgpt.com))

# 0) Prereqs (run on ****every**** monitored cluster)

Enable **User Workload Monitoring** so Prometheus scrapes your app namespaces.

# cluster-monitoring-config.yaml

# Enables user workload monitoring (UWM) so app namespaces are scraped.

apiVersion: v1

kind: ConfigMap

metadata:

name: cluster-monitoring-config

namespace: openshift-monitoring

data:

config.yaml: |

enableUserWorkload: true

---

# user-workload-config.yaml

# (Optional) UWM tuning placeholder. Keep {} if you don't need custom tuning yet.

apiVersion: v1

kind: ConfigMap

metadata:

name: user-workload-config

namespace: openshift-user-workload-monitoring

data:

config.yaml: |

{}

oc apply -f cluster-monitoring-config.yaml

oc apply -f user-workload-config.yaml

(Official docs: enabling and configuring UWM.) ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.17/html/monitoring/configuring-user-workload-monitoring?utm_source=chatgpt.com))

# 1) Expose Thanos ****tenancy**** (9092) via a Route (once per cluster)

Do this in **cluster1** and **cluster4** (and repeat for the other clusters you plan to connect later).

# thanos-tenancy-route.yaml

# Purpose: Expose Thanos Querier's "tenancy" port (9092) externally via a secure Route.

apiVersion: route.openshift.io/v1

kind: Route

metadata:

name: thanos-querier-tenancy

namespace: openshift-monitoring

annotations:

# Optional: allow long-running queries through the default router

haproxy.router.openshift.io/timeout: 5m

spec:

# CHANGE ME: Set to your cluster's apps domain

host: thanos-tenancy.apps.cluster1.example.com

to:

kind: Service

name: thanos-querier # The built-in Thanos Querier Service

port:

targetPort: tenancy # <-- selects port 9092 on the Service

tls:

termination: reencrypt

insecureEdgeTerminationPolicy: Redirect

# cluster1

oc -n openshift-monitoring apply -f thanos-querier-tenancy.yaml

# Verify host value

oc -n openshift-monitoring get route thanos-querier-tenancy -o wide

Repeat in **cluster4**, changing spec.host to something like thanos-tenancy.apps.cluster4.example.com.

Why 9092? Port **9092** is guarded by **kube-rbac-proxy** and honors namespace RBAC with namespace=<ns> in the query; **9091** requires cluster-wide perms and returns all namespaces. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

# 2) Namespace-scoped RBAC & service accounts (one per namespace you expose)

## cluster1 → namespace abc1

# cluster1-abc1-rbac.yaml

# Purpose: Create an SA only for querying abc1 metrics via Thanos tenancy;

# bind built-in "view" \*in this namespace only\*.

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos # You asked for this name specifically

namespace: abc1

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: abc1

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: abc1

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole # Bind the built-in ClusterRole...

name: view # ...but only scoped within this namespace via RoleBinding

# cluster1

oc apply -f cluster1-abc1-rbac.yaml

# Short-lived bearer token for Grafana datasource (rotate as needed)

oc -n abc1 create token grafana-thanos --duration=24h > abc1.cluster1.token

## cluster4 → namespace xyz

# cluster4-xyz-rbac.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos

namespace: xyz

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: xyz

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: xyz

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

# cluster4

oc apply -f cluster4-xyz-rbac.yaml

oc -n xyz create token grafana-thanos --duration=24h > xyz.cluster4.token

This matches the blogs’ guidance: **remove cluster-wide cluster-monitoring-view**, bind **view in the project**, and target **9092** with **namespace=<ns>** so queries are constrained to the namespace. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

# 3) Grafana (Helm on cluster3): add ****one datasource per (cluster,namespace)****

You can provision these via Helm values (works reliably vs. some older operator CRD quirks around custom query params). The essentials:

* **URL** → the **9092** Route of the target cluster.
* **Custom query parameters** → namespace=<ns>.
* **Auth** → HTTP header Authorization: Bearer <token>.
* **TLS** → trust the route’s cert; set tlsSkipVerify: false if you install proper certs (or true temporarily if you must).

# values.yaml (excerpt) for your Grafana Helm release in cluster3

grafana:

additionalDataSources:

# ---------- cluster1 / abc1 ----------

- name: ocp-cluster1-abc1

type: prometheus

url: https://thanos-tenancy.apps.cluster1.example.com

access: proxy

jsonData:

# CRITICAL: enforce namespace tenancy at the datasource layer

customQueryParameters: "namespace=abc1"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false # Prefer proper trust instead of skipping

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER1\_ABC1\_TOKEN}"

# ---------- cluster4 / xyz ----------

- name: ocp-cluster4-xyz

type: prometheus

url: https://thanos-tenancy.apps.cluster4.example.com

access: proxy

jsonData:

customQueryParameters: "namespace=xyz"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER4\_XYZ\_TOKEN}"

How to pass the tokens:

* Create Kubernetes Secrets in the Grafana namespace (cluster3) and map them to env or provision them via Helm --set-file. Example:

# Create secrets from the token files you exported earlier

kubectl -n grafana create secret generic cluster1-abc1-ds --from-file=token=abc1.cluster1.token

kubectl -n grafana create secret generic cluster4-xyz-ds --from-file=token=xyz.cluster4.token

Then reference them as env and expand into your provisioning (one common pattern is initContainer to write the token into a file Grafana can read, or Helm templating to inject token values directly at install time). The exact wiring depends on your Helm chart variant; the key point is that Grafana sends the **Bearer** header.

The **namespace** custom query param is required by the 9092 tenancy endpoint; Grafana won’t auto-add it, so set it explicitly. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/), [Red Hat](https://www.redhat.com/tracks/_pfcdn/assets/10330/contents/548954/4c855e59-fd2a-49f8-8b84-76b31ccb2ef7.pdf?utm_source=chatgpt.com))

# 4) (Optional but recommended) TLS trust & ingress restrictions

* If your Routes use a private CA, **mount the trust bundle** into Grafana and set tlsSkipVerify: false.
* You can add **allowlists/mTLS** or use a dedicated IngressController so only Grafana (cluster3) can reach each cluster’s 9092 Route.

(These align with the OAuth/trust guidance in the community/Red Hat posts.) ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

# 5) Test your isolation

In Grafana, select **ocp-cluster1-abc1** and run:

sum(rate(container\_cpu\_usage\_seconds\_total{namespace="abc1"}[5m])) by (pod)

Then switch to **ocp-cluster4-xyz** and run the same query with namespace="xyz". You should only see metrics from the chosen namespace; cross-namespace series won’t appear because the **tenancy layer** filters them server-side. (The blog demonstrates this exact behavior and the reason 9091 showed all projects.) ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

## Is a single Grafana in cluster3 OK for all 10 clusters?

**Yes.** Requirements:

* **Network reachability** from cluster3 → each cluster’s **9092** Route.
* **TLS trust** for those Routes in Grafana.
* **One datasource per (cluster,namespace)** each with its **own token**.
* **Capacity**: size Grafana to handle dashboard concurrency (CPU/mem, database).  
  (Thanos tenancy endpoint is explicitly intended for this use.) ([Red Hat](https://www.redhat.com/tracks/_pfcdn/assets/10330/contents/548954/4c855e59-fd2a-49f8-8b84-76b31ccb2ef7.pdf?utm_source=chatgpt.com))

## Quick copy-paste pack (your two namespaces)

### cluster1

# 01-route.yaml (cluster1)

apiVersion: route.openshift.io/v1

kind: Route

metadata:

name: thanos-querier-tenancy

namespace: openshift-monitoring

spec:

host: thanos-tenancy.apps.cluster1.example.com

to:

kind: Service

name: thanos-querier

port:

targetPort: tenancy

tls:

termination: reencrypt

insecureEdgeTerminationPolicy: Redirect

---

# 02-rbac-abc1.yaml (cluster1)

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos

namespace: abc1

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: abc1

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: abc1

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

oc -n openshift-monitoring apply -f 01-route.yaml

oc apply -f 02-rbac-abc1.yaml

oc -n abc1 create token grafana-thanos --duration=24h > abc1.cluster1.token

### cluster4

# 01-route.yaml (cluster4)

apiVersion: route.openshift.io/v1

kind: Route

metadata:

name: thanos-querier-tenancy

namespace: openshift-monitoring

spec:

host: thanos-tenancy.apps.cluster4.example.com

to:

kind: Service

name: thanos-querier

port:

targetPort: tenancy

tls:

termination: reencrypt

insecureEdgeTerminationPolicy: Redirect

---

# 02-rbac-xyz.yaml (cluster4)

apiVersion: v1

kind: ServiceAccount

metadata:

name: grafana-thanos

namespace: xyz

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-view

namespace: xyz

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: xyz

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

oc -n openshift-monitoring apply -f 01-route.yaml

oc apply -f 02-rbac-xyz.yaml

oc -n xyz create token grafana-thanos --duration=24h > xyz.cluster4.token

### cluster3 (Grafana Helm values excerpt)

grafana:

additionalDataSources:

- name: ocp-cluster1-abc1

type: prometheus

url: https://thanos-tenancy.apps.cluster1.example.com

access: proxy

jsonData:

customQueryParameters: "namespace=abc1"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER1\_ABC1\_TOKEN}"

- name: ocp-cluster4-xyz

type: prometheus

url: https://thanos-tenancy.apps.cluster4.example.com

access: proxy

jsonData:

customQueryParameters: "namespace=xyz"

httpHeaderName1: "Authorization"

timeInterval: "5s"

tlsSkipVerify: false

secureJsonData:

httpHeaderValue1: "Bearer ${CLUSTER4\_XYZ\_TOKEN}"

If you ever see “all namespaces” again, verify you didn’t accidentally point at **9091** or grant cluster-monitoring-view. The fix is to use **9092** + per-namespace view + namespace= param. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))

## References you asked me to use

* **Community blog (deep dive with screenshots & exact steps)** showing why **9091** leaks all namespaces and how **9092** + namespace= + namespace-scoped view fixes it. ([TechBlog](https://blog.stderr.at/openshift/2020/12/thanos-querier-vs-thanos-querier/))
* **Red Hat blog/slide content** reinforcing the same pattern (9092 tenancy, namespace= param, SA needs just view in namespace; example Grafana datasource fields). ([Red Hat](https://www.redhat.com/en/blog/thanos-querier-versus-thanos-querier?utm_source=chatgpt.com))
* **OpenShift docs** for enabling UWM and accessing metrics concepts. ([Red Hat Docs](https://docs.redhat.com/en/documentation/openshift_container_platform/4.17/html/monitoring/configuring-user-workload-monitoring?utm_source=chatgpt.com))

If you share the **apps domains** for all 10 clusters and the list of **(cluster, namespace)** pairs, I can spit out a single bundled set of manifests + a ready-to-merge Helm values.yaml with all datasources.

Good question 👍 — since you want **namespace-scoped access** to Thanos metrics, you don’t need to bind a wide ClusterRole with cluster-wide permissions (which would break your isolation goal).

But, there are some gotchas with OpenShift Monitoring:

* Thanos Querier itself runs in openshift-monitoring (or openshift-user-workload-monitoring).
* Its API does not natively filter per namespace, but you can restrict what **service account tokens** can query by scoping them to read only metrics that are labeled with their namespace.
* Instead of a **ClusterRole**, you should create a **Role** in the target namespace and bind it to your grafana-thanos service account.

However, if you really do need a **ClusterRole** (for example, if Thanos requires access across namespaces but you want to narrow permissions by label selectors), then here’s a minimal example:

**Example: ClusterRole (only if absolutely required)**

# This ClusterRole grants GET access to metrics via Thanos Querier

# Use carefully, since this is cluster-scoped and could expose all metrics

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: grafana-thanos-read

rules:

- apiGroups: [""]

resources:

- pods

- services

- endpoints

verbs: ["get", "list", "watch"]

- nonResourceURLs: ["/api", "/api/\*", "/metrics", "/federate", "/api/v1/query", "/api/v1/query\_range"]

verbs: ["get"]

**Example: ClusterRoleBinding (to your SA)**

# This binds the ClusterRole to the grafana-thanos service account in a namespace

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: grafana-thanos-read-binding

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: abc1 # <-- change per namespace (e.g., abc1, xyz, etc.)

roleRef:

kind: ClusterRole

name: grafana-thanos-read

apiGroup: rbac.authorization.k8s.io

**Best Practice (Recommended)**

Instead of the above ClusterRole, **use namespace-scoped Role** like this:

# Role scoped only to a single namespace

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: grafana-thanos-namespace-read

namespace: abc1 # only abc1 namespace

rules:

- apiGroups: [""]

resources:

- pods

- services

- endpoints

verbs: ["get", "list", "watch"]

And bind it:

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: grafana-thanos-namespace-binding

namespace: abc1

subjects:

- kind: ServiceAccount

name: grafana-thanos

namespace: abc1

roleRef:

kind: Role

name: grafana-thanos-namespace-read

apiGroup: rbac.authorization.k8s.io

✅ **Answering your question directly:**

* Use **Role/RoleBinding** per namespace (abc1, xyz, etc.) for isolation.
* Only use a **ClusterRole/ClusterRoleBinding** if Thanos/Prometheus enforces access only via cluster-wide rules (not typical for namespace-isolation setups).

👉 Do you want me to generate the **full working YAML set** for both namespaces (abc1 in cluster1, xyz in cluster4), including the **Grafana datasource secrets** pointing to Thanos with those service accounts? That way you’ll have end-to-end manifests.