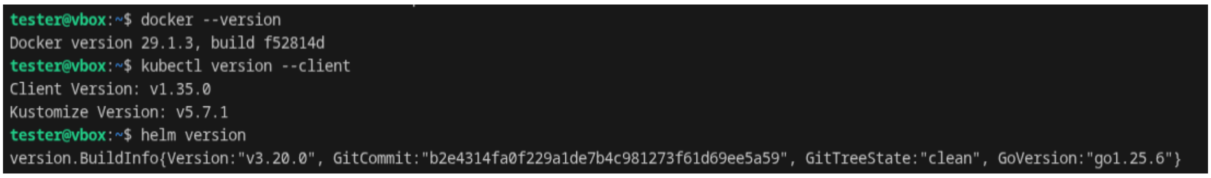
Verify Docker installation.

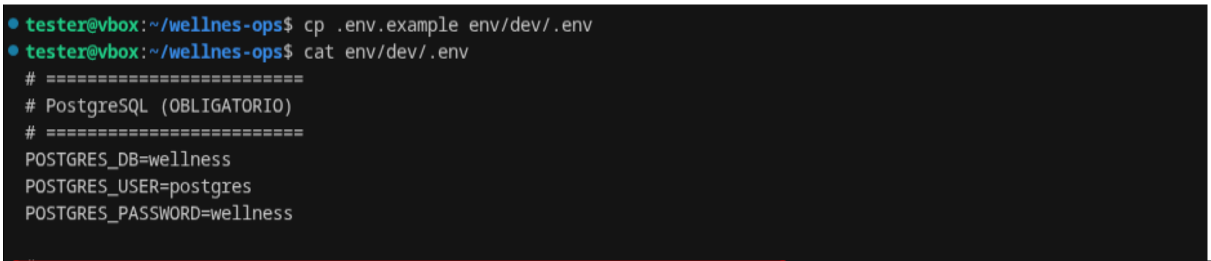
In this setup we are running Docker version 29.1.3, but any recent stable release works.



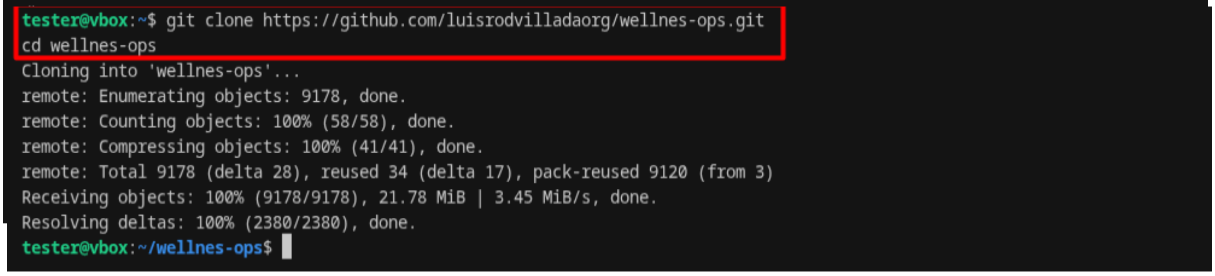
```
tester@vbox:~$ docker --version
Docker version 29.1.3, build f52814d
tester@vbox:~$ kubectl version --client
Client Version: v1.35.0
Kustomize Version: v5.7.1
tester@vbox:~$ helm version
version.BuildInfo{Version:"v3.20.0", GitCommit:"b2e4314fa0f229a1de7b4c981273f61d69ee5a59", GitTreeState:"clean", GoVersion:"go1.25.6"}
```

Set the required environment variables for PostgreSQL and the Backend.

The repository includes a `.env.example` file that can be copied and used as a template for local development.



```
tester@vbox:~/wellnes-ops$ cp .env.example env/dev/.env
tester@vbox:~/wellnes-ops$ cat env/dev/.env
 # ========================
 # PostgreSQL (OBLIGATORIO)
 # ========================
 POSTGRES_DB=wellness
 POSTGRES_USER=postgres
 POSTGRES_PASSWORD=wellness
```

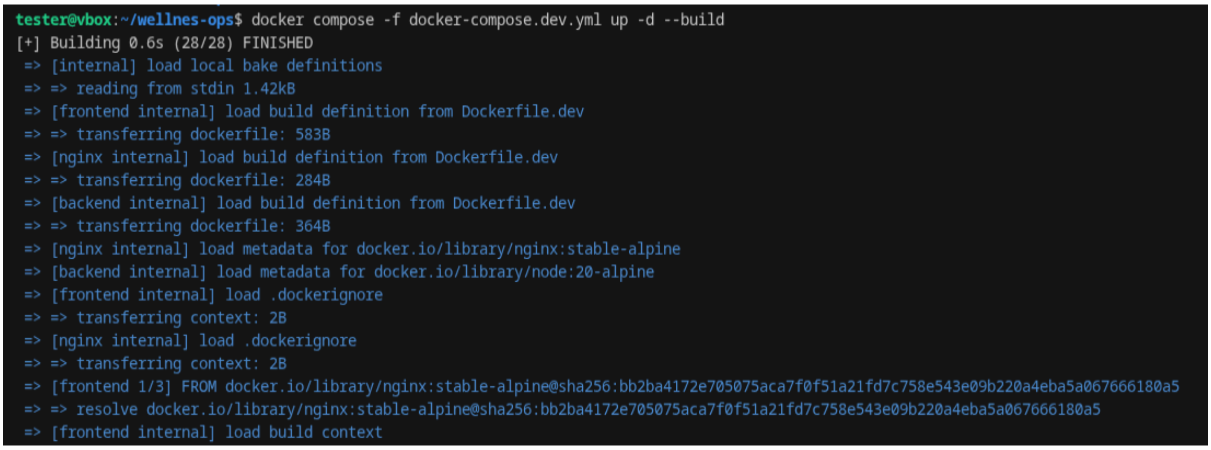


```
tester@vbox:~$ git clone https://github.com/luisrodvilladaorg/wellnes-ops.git
cd wellnes-ops
Cloning into 'wellnes-ops'...
remote: Enumerating objects: 9178, done.
remote: Counting objects: 100% (58/58), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 9178 (delta 28), reused 34 (delta 17), pack-reused 9120 (from 3)
Receiving objects: 100% (9178/9178), 21.78 MiB | 3.45 MiB/s, done.
Resolving deltas: 100% (2380/2380), done.
tester@vbox:~/wellnes-ops$
```

Start the container stack using Docker Compose.

In this example we use the `dev` environment, but the `prod` environment can be used the same way depending on the deployment scenario.

docker compose -f docker-compose-dev up -d --build

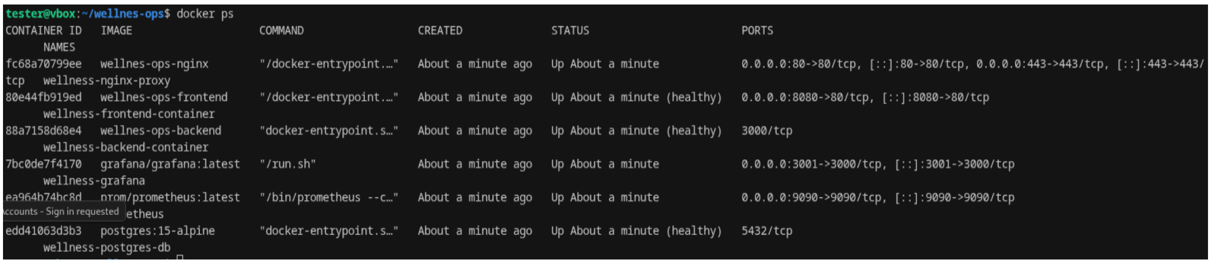


```
tester@vbox:~/wellnes-ops$ docker compose -f docker-compose.dev.yml up -d --build
[+] Building 0.6s (28/28) FINISHED
 => [internal] load local bake definitions
 => => reading from stdin 1.42kB
 => [frontend internal] load build definition from Dockerfile.dev
 => => transferring dockerfile: 583B
 => [nginx internal] load build definition from Dockerfile.dev
 => => transferring dockerfile: 284B
 => [backend internal] load build definition from Dockerfile.dev
 => => transferring dockerfile: 364B
 => [nginx internal] load metadata for docker.io/library/nginx:stable-alpine
 => [backend internal] load metadata for docker.io/library/node:20-alpine
 => [frontend internal] load .dockerignore
 => => transferring context: 2B
 => [nginx internal] load .dockerignore
 => => transferring context: 2B
 => [frontend 1/3] FROM docker.io/library/nginx:stable-alpine@sha256:bb2ba4172e705075aca7f0f51a21fd7c758e543e09b220a4eba5a067666180a5
 => => resolve docker.io/library/nginx:stable-alpine@sha256:bb2ba4172e705075aca7f0f51a21fd7c758e543e09b220a4eba5a067666180a5
 => [frontend internal] load build context
```

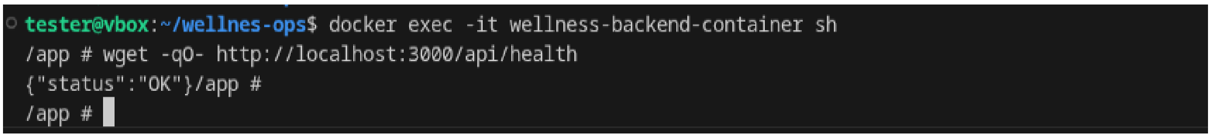
For security reasons, the Backend service is not exposed externally.

To validate connectivity, exec into the Backend container and test the internal endpoint directly.

```
docker exec -it wellness-backend-container sh
wget -q0- http://localhost:3000/api/health
```



```
tester@vbox:~/wellnes-ops$ docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED              STATUS                        PORTS
      NAMES
fc68a70799ee   wellnes-ops-nginx         "/docker-entrypoint.…"   About a minute ago   Up About a minute             0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/
tcp   wellness-nginx-proxy
80e44fb919ed   wellnes-ops-frontend      "/docker-entrypoint.…"   About a minute ago   Up About a minute (healthy)   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
      wellness-frontend-container
88a7158d68e4   wellnes-ops-backend       "docker-entrypoint.s…"   About a minute ago   Up About a minute (healthy)   3000/tcp
      wellness-backend-container
7bc0de7f4170   grafana/grafana:latest    "/run.sh"                About a minute ago   Up About a minute             0.0.0.0:3001->3000/tcp, [::]:3001->3000/tcp
      wellness-grafana
ea964b74bc8d   prom/prometheus:latest    "/bin/prometheus --c…"   About a minute ago   Up About a minute             0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp
ccounts - Sign in requested etheus
edd41063d3b3   postgres:15-alpine        "docker-entrypoint.s…"   About a minute ago   Up About a minute (healthy)   5432/tcp
      wellness-postgres-db
```

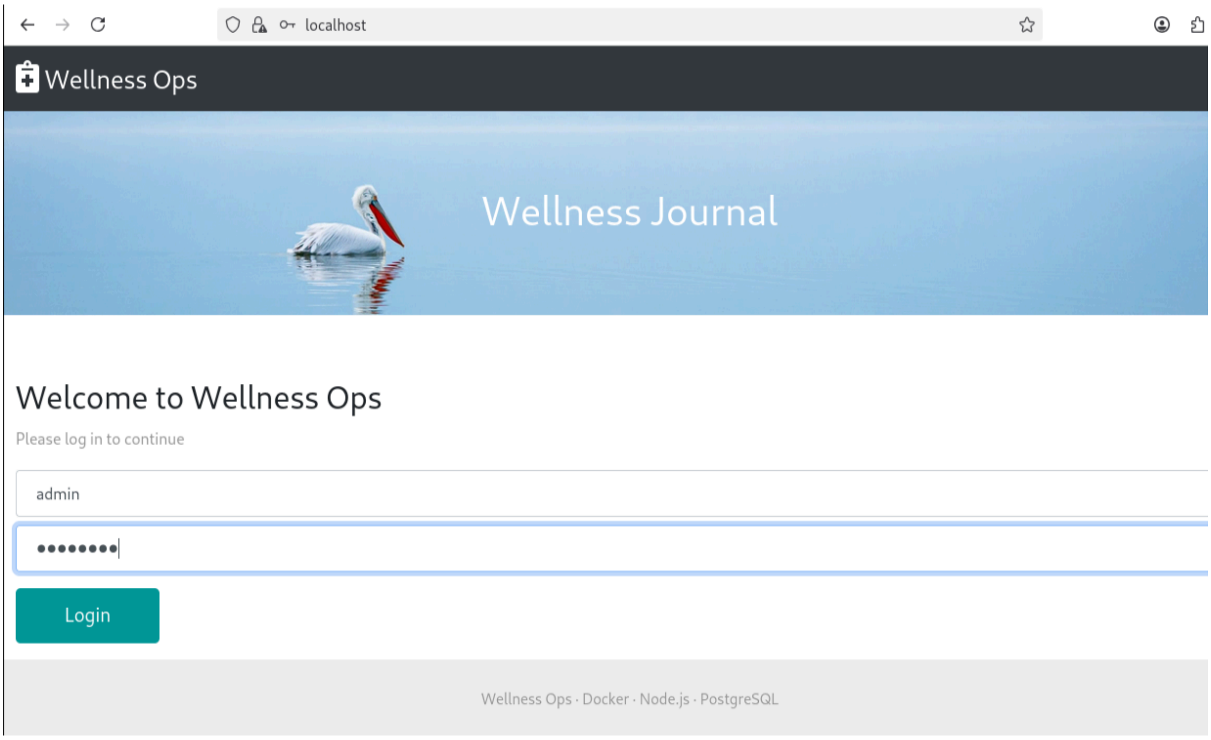


```
tester@vbox:~/wellnes-ops$ docker exec -it wellness-backend-container sh
/app # wget -q0- http://localhost:3000/api/health
{"status":"OK"}/app #
/app #
```

Access the application through any web browser using TLS:

https://localhost

Log in using the default development credentials: admin / admin123



Wellness Ops

Wellness Journal

Welcome to Wellness Ops

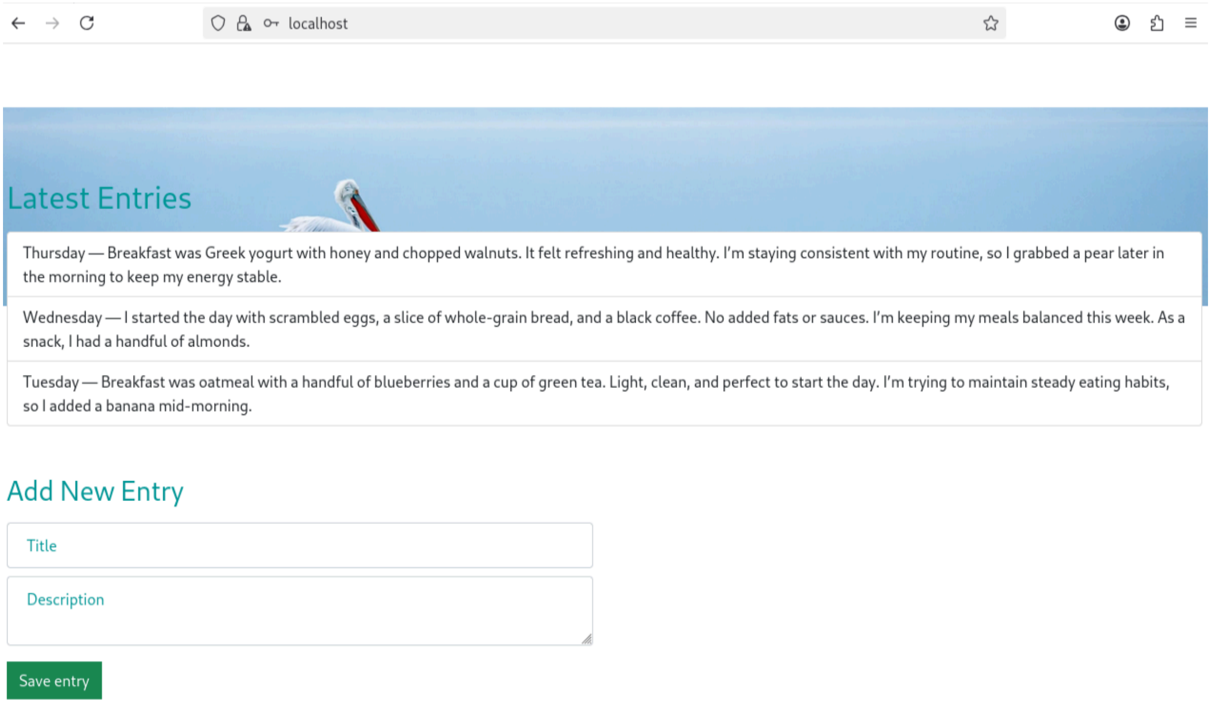
Please log in to continue

admin

••••••••

Wellness Ops · Docker · Node.js · PostgreSQL

Create and save a few records to validate that data persistence works as expected.



Latest Entries

Thursday — Breakfast was Greek yogurt with honey and chopped walnuts. It felt refreshing and healthy. I'm staying consistent with my routine, so I grabbed a pear later in the morning to keep my energy stable.

Wednesday — I started the day with scrambled eggs, a slice of whole-grain bread, and a black coffee. No added fats or sauces. I'm keeping my meals balanced this week. As a snack, I had a handful of almonds.

Tuesday — Breakfast was oatmeal with a handful of blueberries and a cup of green tea. Light, clean, and perfect to start the day. I'm trying to maintain steady eating habits, so I added a banana mid-morning.

Add New Entry

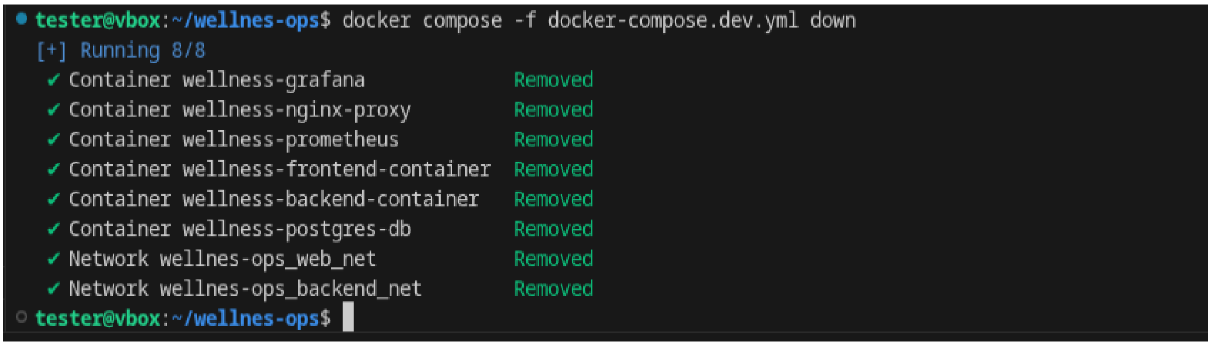
Title

Description

Save entry

Once the application is confirmed to be running correctly, shut it down before proceeding with the next steps..

docker compose -f docker-compose.dev-yml down

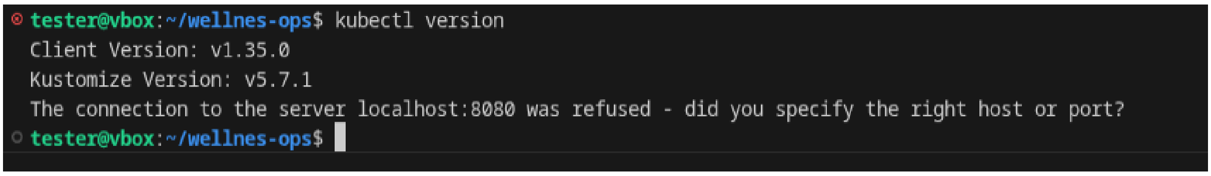


```
● tester@vbox:~/wellnes-ops$ docker compose -f docker-compose.dev.yml down
  [+] Running 8/8
   ✔ Container wellness-grafana            Removed
   ✔ Container wellness-nginx-proxy        Removed
   ✔ Container wellness-prometheus         Removed
   ✔ Container wellness-frontend-container Removed
   ✔ Container wellness-backend-container  Removed
   ✔ Container wellness-postgres-db        Removed
   ✔ Network wellnes-ops_web_net           Removed
   ✔ Network wellnes-ops_backend_net       Removed
○ tester@vbox:~/wellnes-ops$ 
```

KUBERNETES

Verify kubectl Installation

Ensure that `kubectl` is installed and accessible from the system. This confirms that the Kubernetes CLI is available before interacting with the cluster.



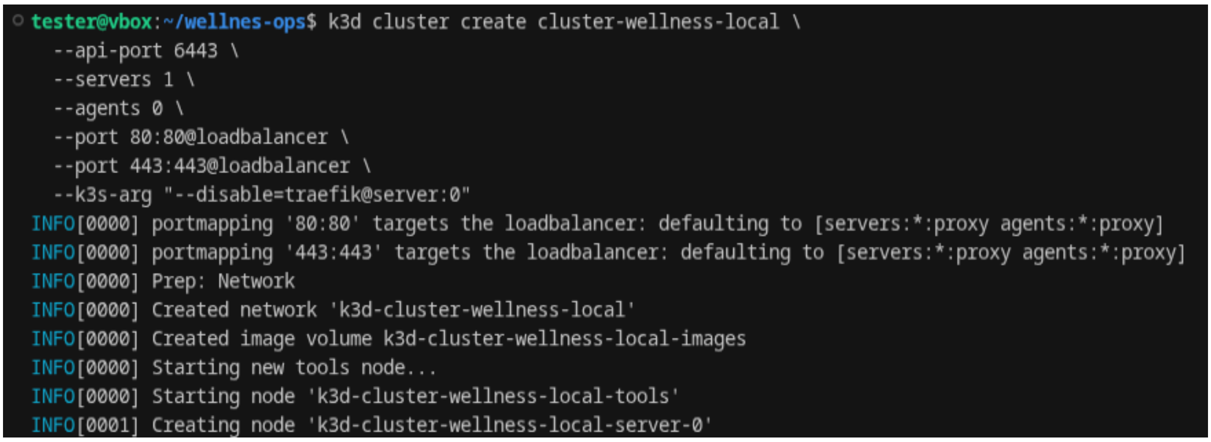
```
tester@vbox:~/wellnes-ops$ kubectl version
Client Version: v1.35.0
Kustomize Version: v5.7.1
The connection to the server localhost:8080 was refused - did you specify the right host or port?
tester@vbox:~/wellnes-ops$
```

Create the Kubernetes cluster

For local environments—such as this setup running Debian 12 inside a VirtualBox VM—we provision a Kubernetes cluster. In this example, the cluster is named *cluster-wellness-local*

Run the following commands to initialize the cluster

```
k3d cluster create cluster-wellness-local \
  --api-port 6443 \
  --servers 1 \
  --agents 0 \
  --port 80:80@loadbalancer \
  --port 443:443@loadbalancer \
  --k3s-arg "--disable=traefik@server:0"
```



```
tester@vbox:~/wellnes-ops$ k3d cluster create cluster-wellness-local \
  --api-port 6443 \
  --servers 1 \
  --agents 0 \
  --port 80:80@loadbalancer \
  --port 443:443@loadbalancer \
  --k3s-arg "--disable=traefik@server:0"
INFO[0000] portmapping '80:80' targets the loadbalancer: defaulting to [servers:*:proxy agents:*:proxy]
INFO[0000] portmapping '443:443' targets the loadbalancer: defaulting to [servers:*:proxy agents:*:proxy]
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-cluster-wellness-local'
INFO[0000] Created image volume k3d-cluster-wellness-local-images
INFO[0000] Starting new tools node...
INFO[0000] Starting node 'k3d-cluster-wellness-local-tools'
INFO[0001] Creating node 'k3d-cluster-wellness-local-server-0'
```

Step by step

Create the Kubernetes configuration directory if it does not already exist:

Extract the cluster kubeconfig and write it to the default configuration file: This ensures that kubectl interacts with the correct cluster context.

Set the appropriate file permissions: Restrict access so only the current user can read the kubeconfig

List available contexts to verify that the configuration was applied correctly**:**

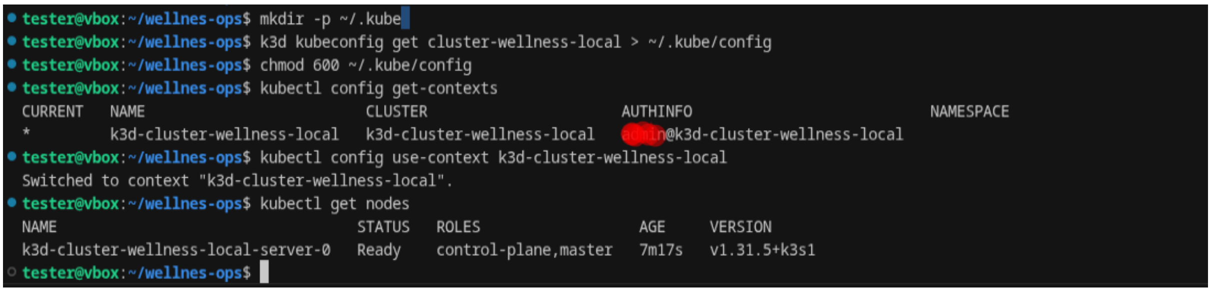
Commands:

mkdir -p ~/.kube

k3d kubeconfig get cluster-wellness-local > ~/.kube/config

chmod 600 ~/.kube/config

kubectl config get-contexts



```
tester@vbox:~/wellnes-ops$ mkdir -p ~/.kube
tester@vbox:~/wellnes-ops$ k3d kubeconfig get cluster-wellness-local > ~/.kube/config
tester@vbox:~/wellnes-ops$ chmod 600 ~/.kube/config
tester@vbox:~/wellnes-ops$ kubectl config get-contexts
CURRENT   NAME                     CLUSTER                  AUTHINFO                            NAMESPACE
*         k3d-cluster-wellness-local   k3d-cluster-wellness-local   [redacted]@k3d-cluster-wellness-local
tester@vbox:~/wellnes-ops$ kubectl config use-context k3d-cluster-wellness-local
Switched to context "k3d-cluster-wellness-local".
tester@vbox:~/wellnes-ops$ kubectl get nodes
NAME                                STATUS   ROLES                  AGE     VERSION
k3d-cluster-wellness-local-server-0   Ready    control-plane,master   7m17s   v1.31.5+k3s1
tester@vbox:~/wellnes-ops$ 
```

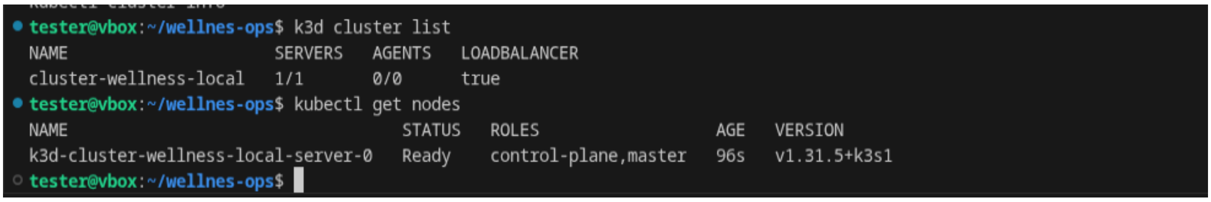
Verify that the Kubernetes cluster is installed and running.

Since this environment uses k3d, confirm that the cluster is active and recognized by the local system

Check that the Kubernetes nodes are registered and in a Ready state

This ensures the control plane and worker nodes are operational.

k3d cluster list
kubectl get nodes

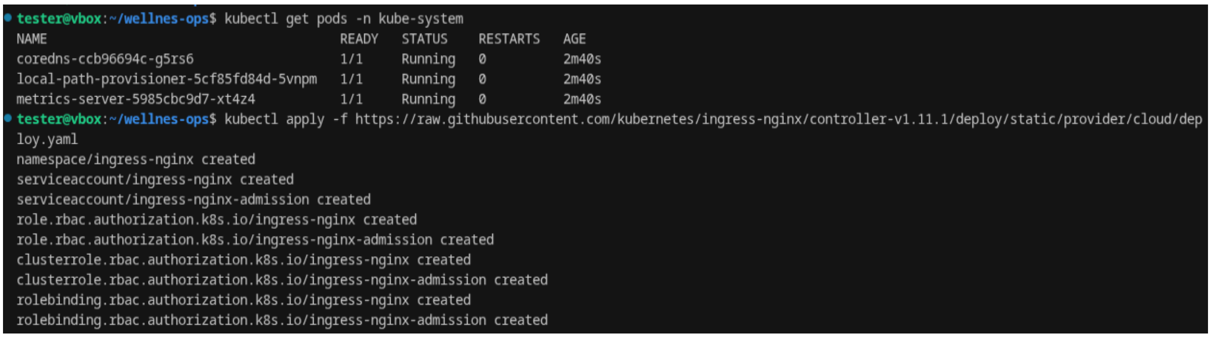


```
● tester@vbox:~/wellnes-ops$ k3d cluster list
  NAME                    SERVERS   AGENTS   LOADBALANCER
  cluster-wellness-local  1/1       0/0      true
● tester@vbox:~/wellnes-ops$ kubectl get nodes
  NAME                             STATUS   ROLES                  AGE   VERSION
  k3d-cluster-wellness-local-server-0   Ready    control-plane,master   96s   v1.31.5+k3s1
○ tester@vbox:~/wellnes-ops$ ▌
```

List the pods running in the `kube-system` namespace.

This allows you to verify that core Kubernetes components (DNS, CNI, metrics, controllers, etc.) are healthy and running as expected.

kubectl get pods -n kube-system



```
● tester@vbox:~/wellnes-ops$ kubectl get pods -n kube-system
  NAME                                     READY   STATUS    RESTARTS   AGE
  coredns-ccb96694c-g5rs6                  1/1     Running   0          2m40s
  local-path-provisioner-5cf85fd84d-5vnpm  1/1     Running   0          2m40s
  metrics-server-5985cbc9d7-xt4z4          1/1     Running   0          2m40s
● tester@vbox:~/wellnes-ops$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.11.1/deploy/static/provider/cloud/dep
  loy.yaml
  namespace/ingress-nginx created
  serviceaccount/ingress-nginx created
  serviceaccount/ingress-nginx-admission created
  role.rbac.authorization.k8s.io/ingress-nginx created
  role.rbac.authorization.k8s.io/ingress-nginx-admission created
  clusterrole.rbac.authorization.k8s.io/ingress-nginx created
  clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
  rolebinding.rbac.authorization.k8s.io/ingress-nginx created
  rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
```

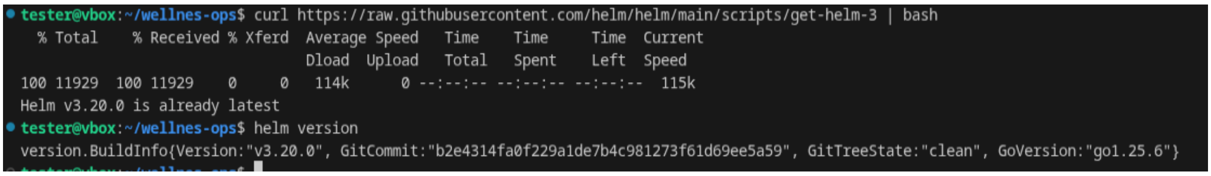
Install Required Tools for the Application

Once the cluster is up and running, install the required tooling for the application stack. This includes Helm, the NGINX Ingress Controller (we replace the default Traefik shipped with k3d), MetalLB, and the MetalLB IP address pool. Finally, apply the necessary Kubernetes manifests.

Install Helm using the official installation script:

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

Verify the installation:



```
tester@vbox:~/wellnes-ops$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11929  100 11929    0     0   114k      0 --:--:-- --:--:-- --:--:--  115k
Helm v3.20.0 is already latest
tester@vbox:~/wellnes-ops$ helm version
version.BuildInfo{Version:"v3.20.0", GitCommit:"b2e4314fa0f229a1de7b4c981273f61d69ee5a59", GitTreeState:"clean", GoVersion:"go1.25.6"}
```

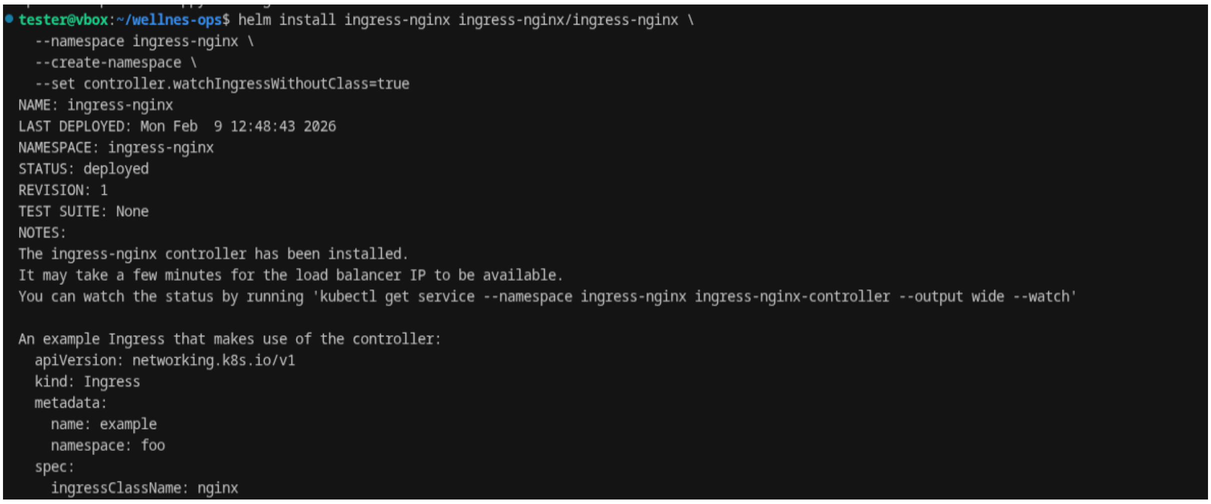
Add the official NGINX Ingress Controller Helm repository and update the index:

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update

Install the NGINX Ingress Controller:

This deployment replaces the default Traefik controller used by k3d and enables NGINX-based ingress routing. The flag `controller.watchIngressWithoutClass=true` allows NGINX to manage ingresses that do not explicitly define an ingressClass

```
helm install ingress-nginx ingress-nginx/ingress-nginx \
  --namespace ingress-nginx \
  --create-namespace \
  --set controller.watchIngressWithoutClass=true
```

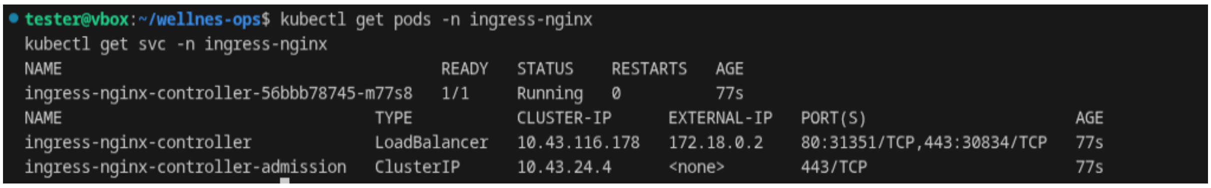


```
tester@vbox:~/wellnes-ops$ helm install ingress-nginx ingress-nginx/ingress-nginx \
  --namespace ingress-nginx \
  --create-namespace \
  --set controller.watchIngressWithoutClass=true
NAME: ingress-nginx
LAST DEPLOYED: Mon Feb  9 12:48:43 2026
NAMESPACE: ingress-nginx
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The ingress-nginx controller has been installed.
It may take a few minutes for the load balancer IP to be available.
You can watch the status by running 'kubectl get service --namespace ingress-nginx ingress-nginx-controller --output wide --watch'

An example Ingress that makes use of the controller:
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: example
    namespace: foo
  spec:
    ingressClassName: nginx
```

Validate NGINX Installation and Deploy MetalLB

This ensures that the ingress layer is operational before deploying any load-balancing components.

```
kubectl get pods -n ingress-nginx
```

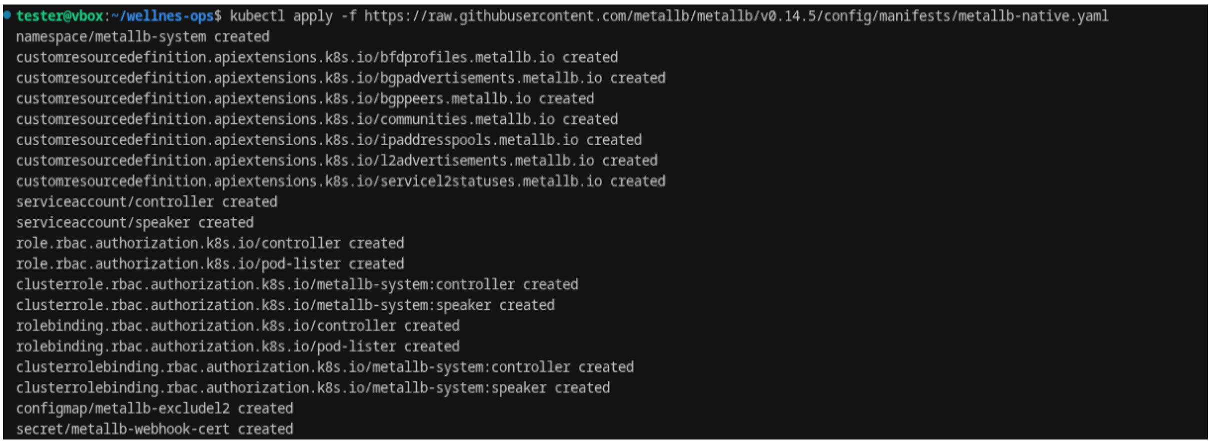


```
tester@vbox:~/wellnes-ops$ kubectl get pods -n ingress-nginx
kubectl get svc -n ingress-nginx
NAME                                       READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-56bbb78745-m77s8  1/1     Running   0          77s
NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller            LoadBalancer   10.43.116.178  172.18.0.2    80:31351/TCP,443:30834/TCP   77s
ingress-nginx-controller-admission  ClusterIP      10.43.24.4     <none>        443/TCP                      77s
```

Install MetalLB using the official manifest.

MetalLB provides Layer 2 load balancing for bare-metal and local Kubernetes clusters, including k3d environments.

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.5/config/manifests/metallb-native.yaml
```



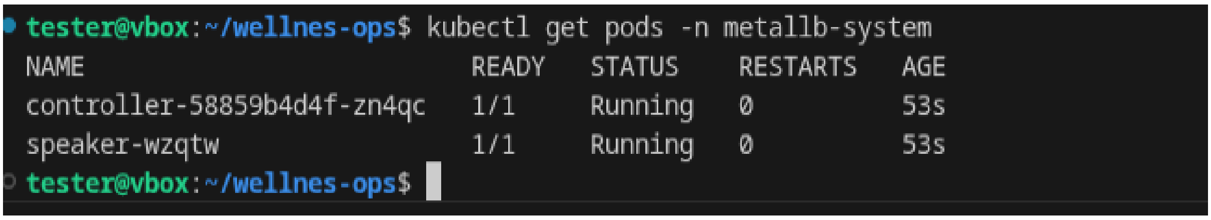
```
tester@vbox:~/wellnes-ops$ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.14.5/config/manifests/metallb-native.yaml
namespace/metallb-system created
customresourcedefinition.apiextensions.k8s.io/bfdprofiles.metallb.io created
customresourcedefinition.apiextensions.k8s.io/bgpadvertisements.metallb.io created
customresourcedefinition.apiextensions.k8s.io/bgppeers.metallb.io created
customresourcedefinition.apiextensions.k8s.io/communities.metallb.io created
customresourcedefinition.apiextensions.k8s.io/ipaddresspools.metallb.io created
customresourcedefinition.apiextensions.k8s.io/l2advertisements.metallb.io created
customresourcedefinition.apiextensions.k8s.io/servicel2statuses.metallb.io created
serviceaccount/controller created
serviceaccount/speaker created
role.rbac.authorization.k8s.io/controller created
role.rbac.authorization.k8s.io/pod-lister created
clusterrole.rbac.authorization.k8s.io/metallb-system:controller created
clusterrole.rbac.authorization.k8s.io/metallb-system:speaker created
rolebinding.rbac.authorization.k8s.io/controller created
rolebinding.rbac.authorization.k8s.io/pod-lister created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:controller created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:speaker created
configmap/metallb-excludel2 created
secret/metallb-webhook-cert created
```

Validate MetalLB and Install IP Address Pool + Prometheus

Verify that MetalLB pods are running correctly:

This confirms that the load balancer components are healthy before configuring the address pool.

kubectl get pods -n metallb-system



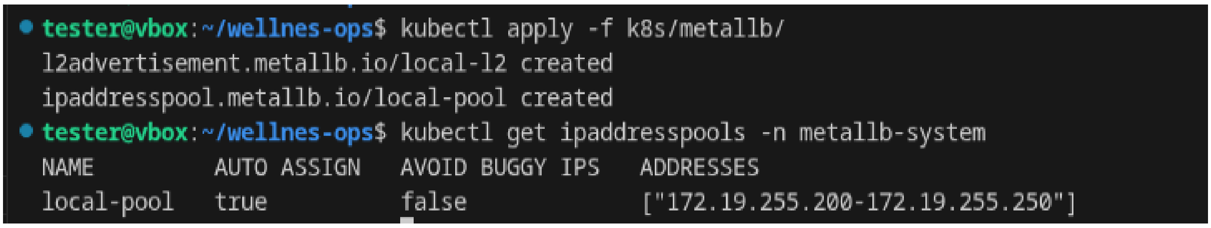
```
tester@vbox:~/wellnes-ops$ kubectl get pods -n metallb-system
NAME                          READY   STATUS    RESTARTS   AGE
controller-58859b4d4f-zn4qc   1/1     Running   0          53s
speaker-wzqtw                 1/1     Running   0          53s
tester@vbox:~/wellnes-ops$
```

Install the MetalLB IP address pool configuration: This defines the range of IPs that MetalLB can allocate for LoadBalancer services.

Validate that the IPAddressPool resource has been created:

kubectl apply -f /k8s/metallb/

kubectl get ipaddresspools -n metallb-system



```
tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/metallb/
l2advertisement.metallb.io/local-l2 created
ipaddresspool.metallb.io/local-pool created
tester@vbox:~/wellnes-ops$ kubectl get ipaddresspools -n metallb-system
NAME         AUTO ASSIGN   AVOID BUGGY IPS   ADDRESSES
local-pool   true          false             ["172.19.255.200-172.19.255.250"]
```

Install Prometheus in an isolated namespace

Add the official Prometheus Helm repository and update the index

Create a dedicated namespace for monitoring components

Install the latest Prometheus chart

kubectl get pods -n monitoring

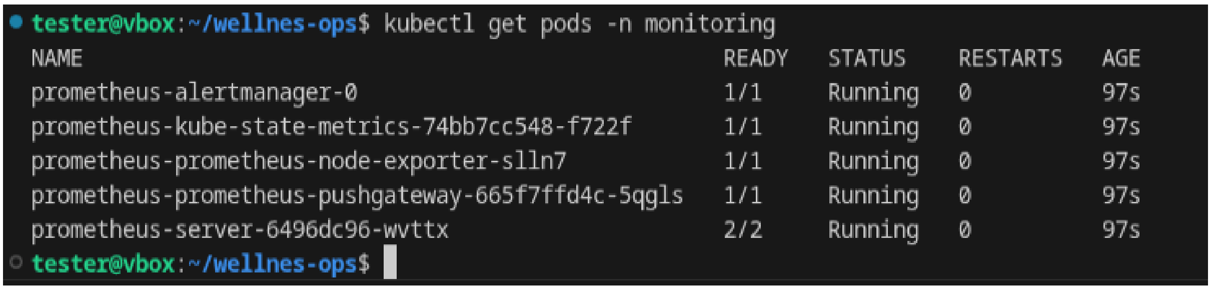
Commands:

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

```
kubectl create namespace monitoring
```

```
helm install prometheus prometheus-community/prometheus -n monitoring
```

```
kubectl get pods -n monitoring
```

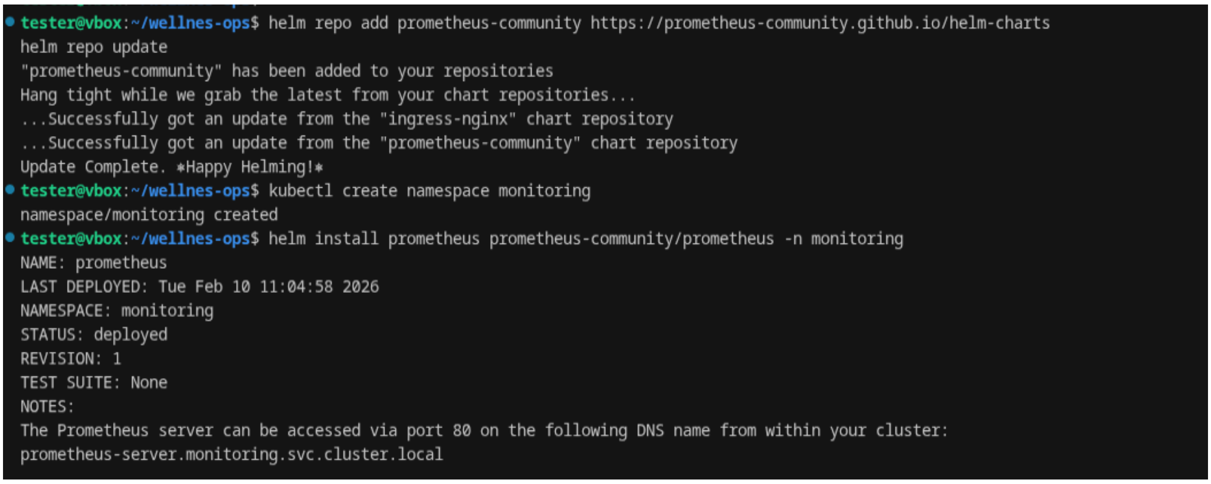


```
tester@vbox:~/wellnes-ops$ kubectl get pods -n monitoring
NAME                                                READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0                           1/1     Running   0          97s
prometheus-kube-state-metrics-74bb7cc548-f722f      1/1     Running   0          97s
prometheus-prometheus-node-exporter-slln7           1/1     Running   0          97s
prometheus-prometheus-pushgateway-665f7ffd4c-5qgls  1/1     Running   0          97s
prometheus-server-6496dc96-wvttx                    2/2     Running   0          97s
tester@vbox:~/wellnes-ops$ 
```

Install Required CRDs

Install the required Custom Resource Definitions (CRDs). These CRDs are needed by the Prometheus stack and must be applied before deploying additional monitoring components.

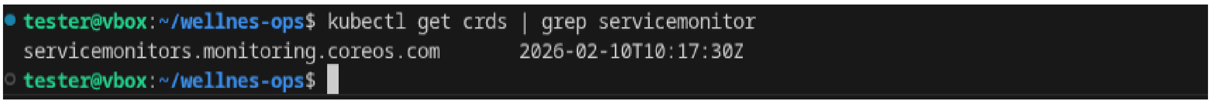
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts



```
tester@vbox:~/wellnes-ops$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
"prometheus-community" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helming!*
tester@vbox:~/wellnes-ops$ kubectl create namespace monitoring
namespace/monitoring created
tester@vbox:~/wellnes-ops$ helm install prometheus prometheus-community/prometheus -n monitoring
NAME: prometheus
LAST DEPLOYED: Tue Feb 10 11:04:58 2026
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-server.monitoring.svc.cluster.local
```

Verify that the ServiceMonitor CRD is installed: This ensures that the monitoring stack can register scrape targets correctly.

kubectl get crds | grep servicemonitor

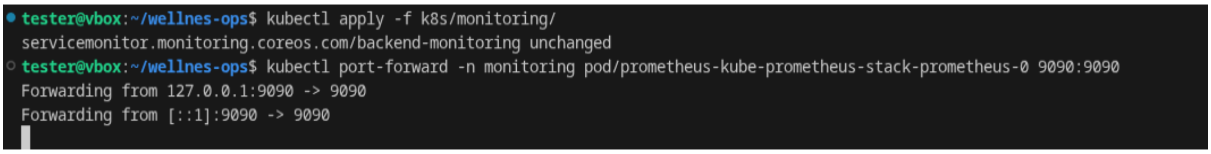


```
tester@vbox:~/wellnes-ops$ kubectl get crds | grep servicemonitor
servicemonitors.monitoring.coreos.com       2026-02-10T10:17:30Z
tester@vbox:~/wellnes-ops$
```

Apply Monitoring Manifests and Expose Prometheus

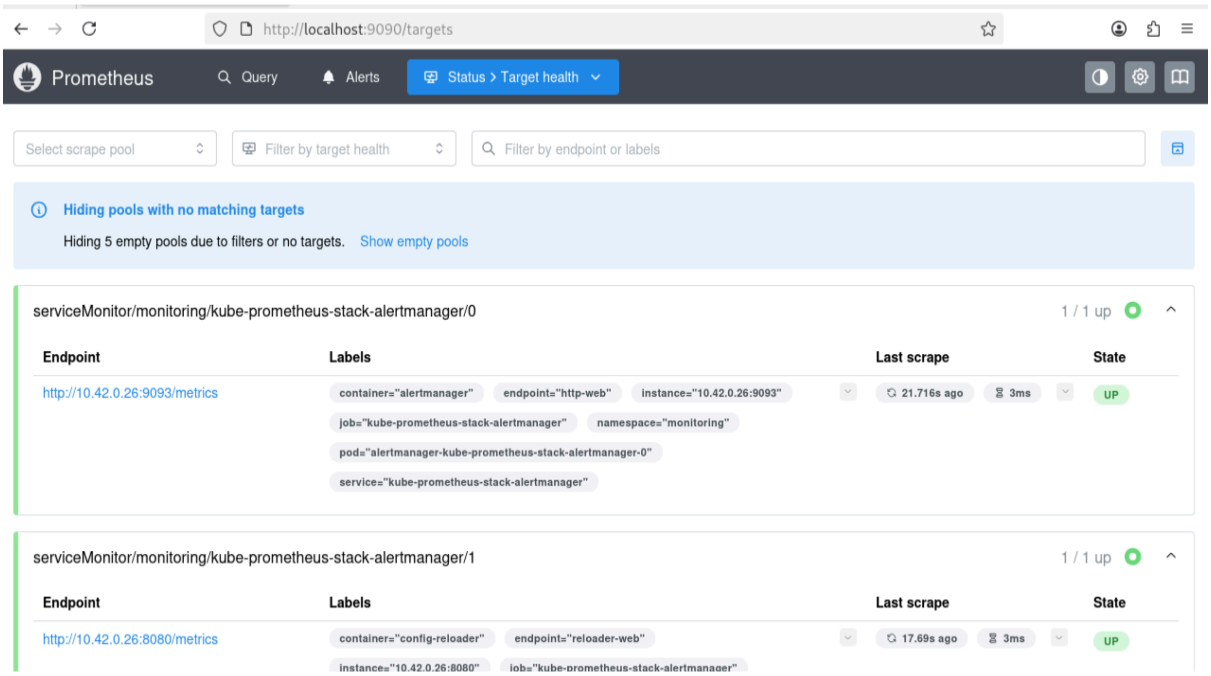
kubectl apply -f k8s/monitoring/

Expose the Prometheus UI locally using port-forwarding: This forwards traffic from port 9090 on your machine to the Prometheus pod.



```
tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/monitoring/
servicemonitor.monitoring.coreos.com/backend-monitoring unchanged
tester@vbox:~/wellnes-ops$ kubectl port-forward -n monitoring pod/prometheus-kube-prometheus-stack-prometheus-0 9090:9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

Validate access and confirm that Prometheus is running (status: Up). Open your browser and navigate to: **http://localhost:9090**



Hiding pools with no matching targets

Hiding 5 empty pools due to filters or no targets. Show empty pools

serviceMonitor/monitoring/kube-prometheus-stack-alertmanager/0 1 / 1 up

Endpoint	Labels	Last scrape	State

http://10.42.0.26:9093/metrics	container="alertmanager"
	endpoint="http-web"
	instance="10.42.0.26:9093"
	job="kube-prometheus-stack-alertmanager"
	namespace="monitoring"
	pod="alertmanager-kube-prometheus-stack-alertmanager-0"
	service="kube-prometheus-stack-alertmanager"
		21.716s ago
		3ms
			UP

serviceMonitor/monitoring/kube-prometheus-stack-alertmanager/1 1 / 1 up

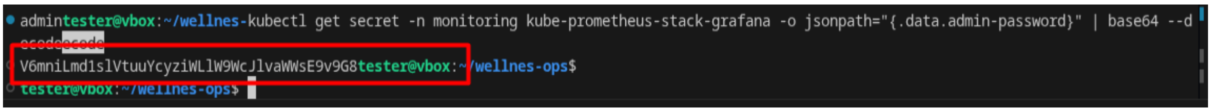
Endpoint	Labels	Last scrape	State

http://10.42.0.26:8080/metrics	container="config-reloader"
	endpoint="reloader-web"
	instance="10.42.0.26:8080"
	job="kube-prometheus-stack-alertmanager"
		17.69s ago
		3ms
			UP

Grafana is included by default in the Prometheus stack

To access the dashboard, retrieve the auto-generated admin password from the Kubernetes secret:

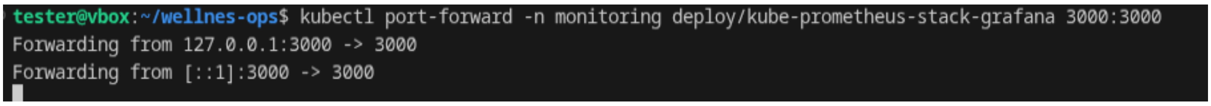
kubectl get secret -n monitoring kube-prometheus-stack-grafana -o jsonpath="{.data.admin-password}" | base64 --decode



```
admintester@vbox:~/wellnes-kubectl get secret -n monitoring kube-prometheus-stack-grafana -o jsonpath="{.data.admin-password}" | base64 --d
ecodeecode
V6mniLmd1slVtuuYcyziWLlW9WcJlvaWWsE9v9G8tester@vbox:~/wellnes-ops$
tester@vbox:~/wellnes-ops$
```

Expose the Grafana dashboard locally using port-forwarding. This allows access to the UI without exposing the service externally.

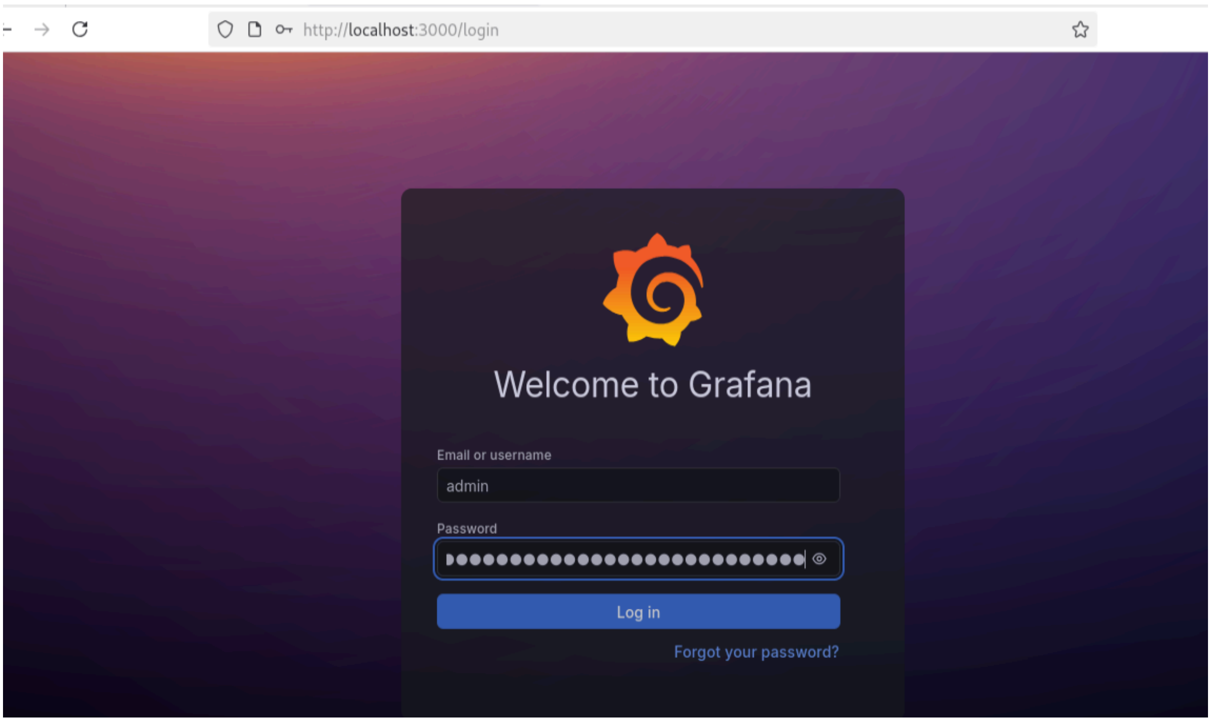
kubectl port-forward -n monitoring deploy/kube-prometheus-stack-grafana 3000:3000



```
tester@vbox:~/wellnes-ops$ kubectl port-forward -n monitoring deploy/kube-prometheus-stack-grafana 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

Access the Grafana dashboard through the local port-forwarding session. Once the port-forward is active, open your browser and navigate to:

http://localhost:3000



http://localhost:3000/login

Welcome to Grafana

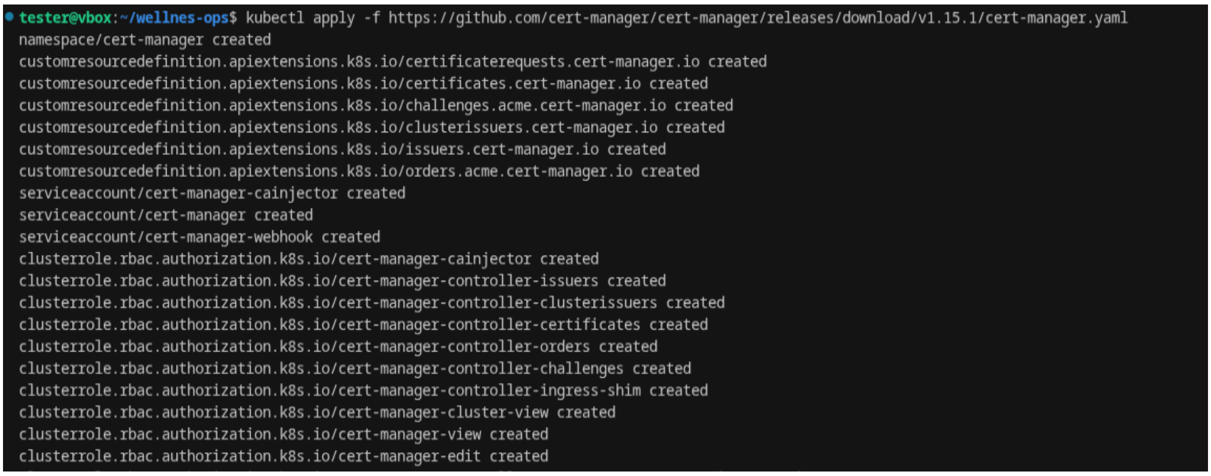
Email or username: admin

Password: ●●●●●●●●●●●●●●●●●●●●●●●●●●●●

Install Cert-Manager

To enable TLS certificate management within the cluster, install Cert-Manager and configure the required certificate resources. Cert-Manager automates the issuance and renewal of TLS certificates for Kubernetes workloads

kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.15.1/cert-manager.yaml

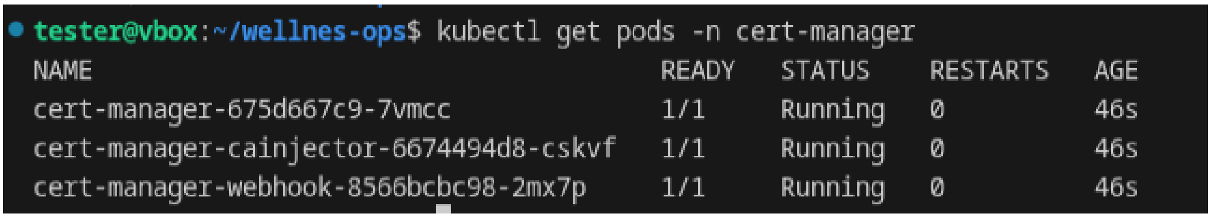


```
tester@vbox:~/wellnes-ops$ kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.15.1/cert-manager.yaml
namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrole.rbac.authorization.k8s.io/cert-manager-cluster-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-edit created
```

Confirm Cert-Manager Pods Are Running

Verify that the Cert-Manager components are installed and running correctly.

This ensures that the certificate management system is fully operational before creating Issuers or Certificates.



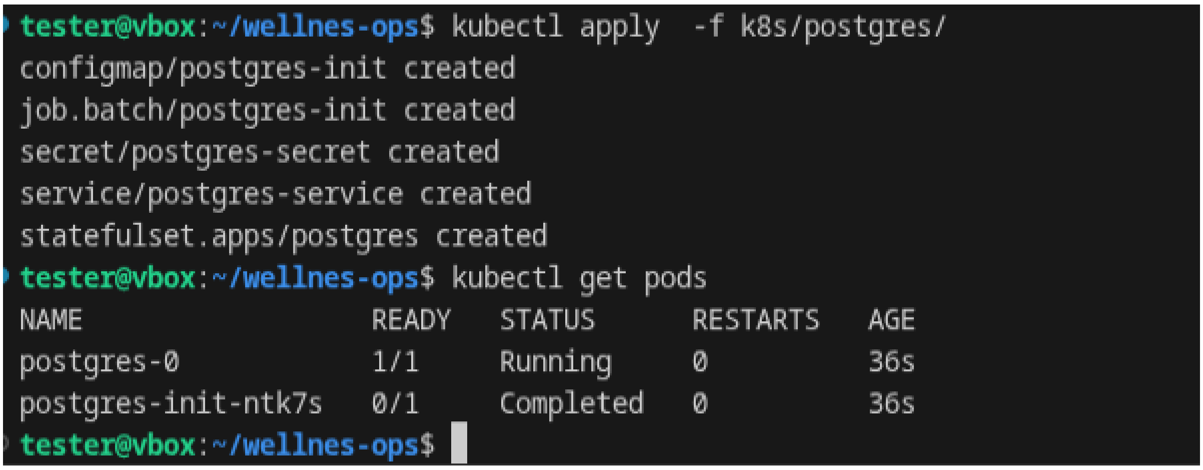
```
tester@vbox:~/wellnes-ops$ kubectl get pods -n cert-manager
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-675d667c9-7vmcc               1/1     Running   0          46s
cert-manager-cainjector-6674494d8-cskvf    1/1     Running   0          46s
cert-manager-webhook-8566bcbc98-2mx7p      1/1     Running   0          46s
```

Apply the application manifests one by one, in the correct order.
It is important to wait ~30 seconds between each deployment to ensure that the pods reach a Running state.

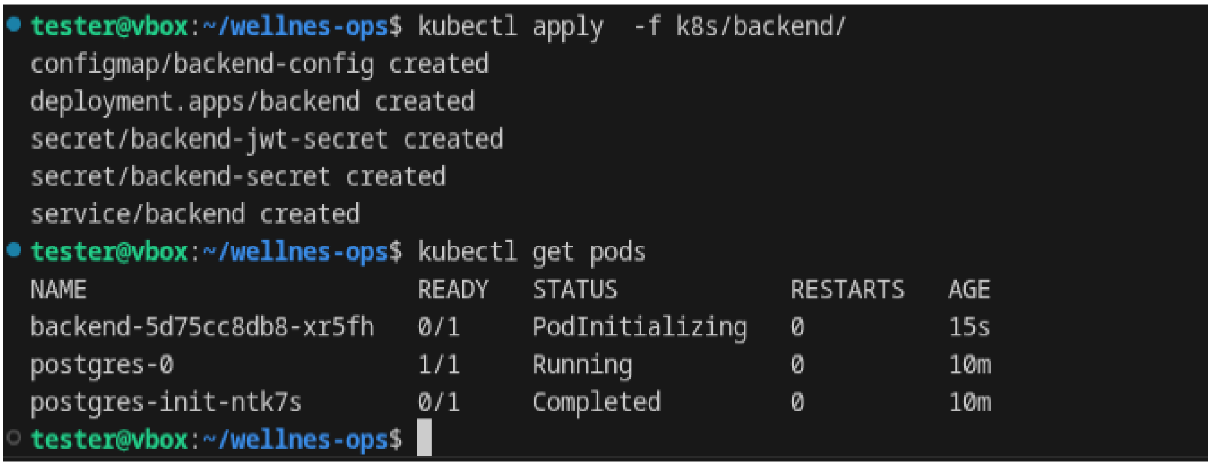
This is required because:

- **The backend depends on the database.**
- **The NGINX ingress depends on the frontend** even if they run in isolated namespaces.

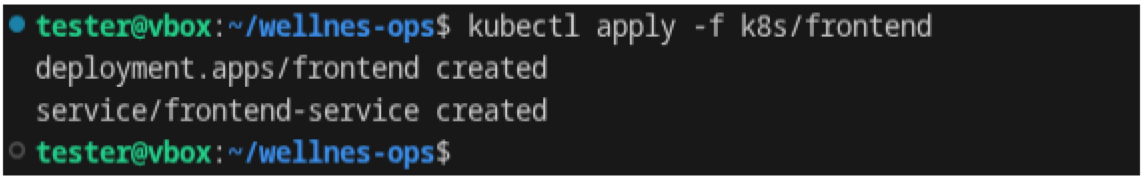
```
kubectl apply  -f /k8s/postgres/
kubectl apply  -f /k8s/backend/
kubectl apply  -f /k8s/frontend/
kubectl apply  -f /k8s/nginx/
```



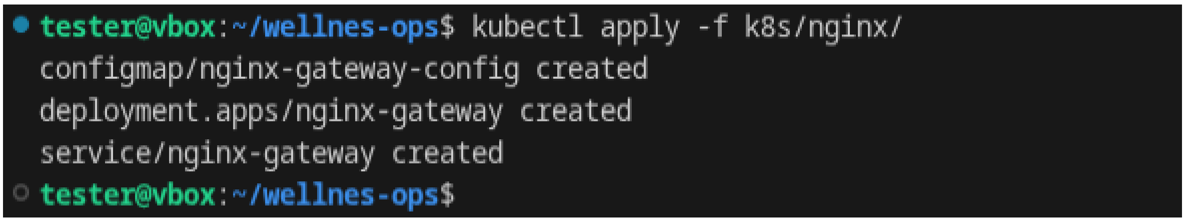
```
tester@vbox:~/wellnes-ops$ kubectl apply  -f k8s/postgres/
configmap/postgres-init created
job.batch/postgres-init created
secret/postgres-secret created
service/postgres-service created
statefulset.apps/postgres created
tester@vbox:~/wellnes-ops$ kubectl get pods
NAME                  READY   STATUS      RESTARTS   AGE
postgres-0            1/1     Running     0          36s
postgres-init-ntk7s   0/1     Completed   0          36s
tester@vbox:~/wellnes-ops$
```



```
tester@vbox:~/wellnes-ops$ kubectl apply  -f k8s/backend/
configmap/backend-config created
deployment.apps/backend created
secret/backend-jwt-secret created
secret/backend-secret created
service/backend created
tester@vbox:~/wellnes-ops$ kubectl get pods
NAME                       READY   STATUS            RESTARTS   AGE
backend-5d75cc8db8-xr5fh   0/1     PodInitializing   0          15s
postgres-0                 1/1     Running           0          10m
postgres-init-ntk7s        0/1     Completed         0          10m
tester@vbox:~/wellnes-ops$
```



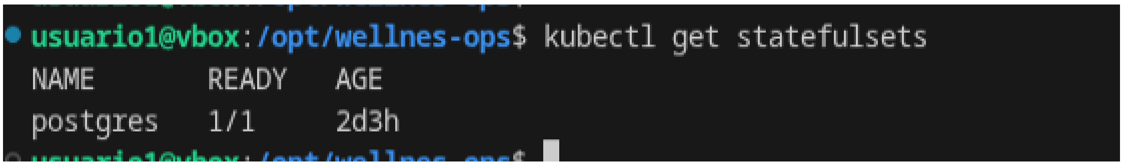
```
tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/frontend
deployment.apps/frontend created
service/frontend-service created
tester@vbox:~/wellnes-ops$
```



```
● tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/nginx/
  configmap/nginx-gateway-config created
  deployment.apps/nginx-gateway created
  service/nginx-gateway created
○ tester@vbox:~/wellnes-ops$
```

Verify PostgreSQL Installation as a StatefulSet

Confirm that PostgreSQL is deployed as a StatefulSet (not just as a Pod). This ensures that the database is using persistent identity and stable storage, as expected for stateful workloads.

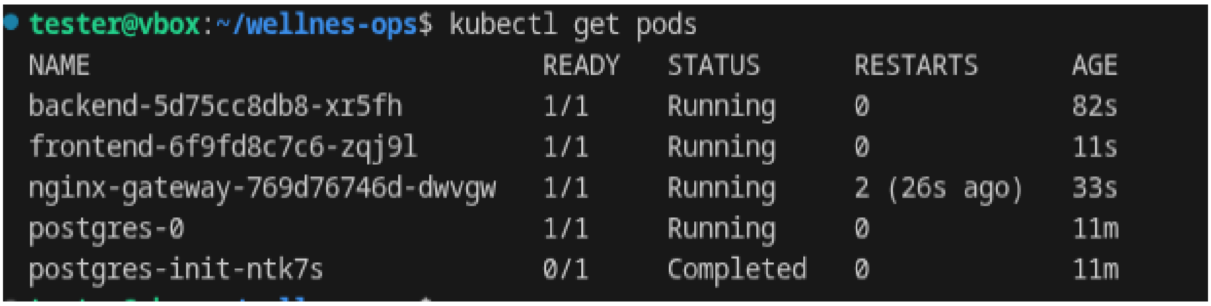


```
● usuario1@vbox:/opt/wellnes-ops$ kubectl get statefulsets
  NAME       READY   AGE
  postgres   1/1     2d3h
○ usuario1@vbox:/opt/wellnes-ops$
```

Verify That All Components Are Running

Confirm that all deployed components are in a Running state.

This ensures that every part of the application stack—database, backend, frontend, and NGINX—is healthy and ready to serve traffic.



```
● tester@vbox:~/wellnes-ops$ kubectl get pods
  NAME                             READY   STATUS      RESTARTS      AGE
  backend-5d75cc8db8-xr5fh         1/1     Running     0             82s
  frontend-6f9fd8c7c6-zqj9l        1/1     Running     0             11s
  nginx-gateway-769d76746d-dwvgw   1/1     Running     2 (26s ago)   33s
  postgres-0                       1/1     Running     0             11m
  postgres-init-ntk7s              0/1     Completed   0             11m
```

Install and Verify Cert-Manager

Install and configure Cert-Manager.

This component is required for automated TLS certificate issuance and renewal inside the cluster.

```
kubectl create namespace cert-manager
helm repo add jetstack https://charts.jetstack.io
helm repo update
```

Install Cert-Manager with CRDs enabled:

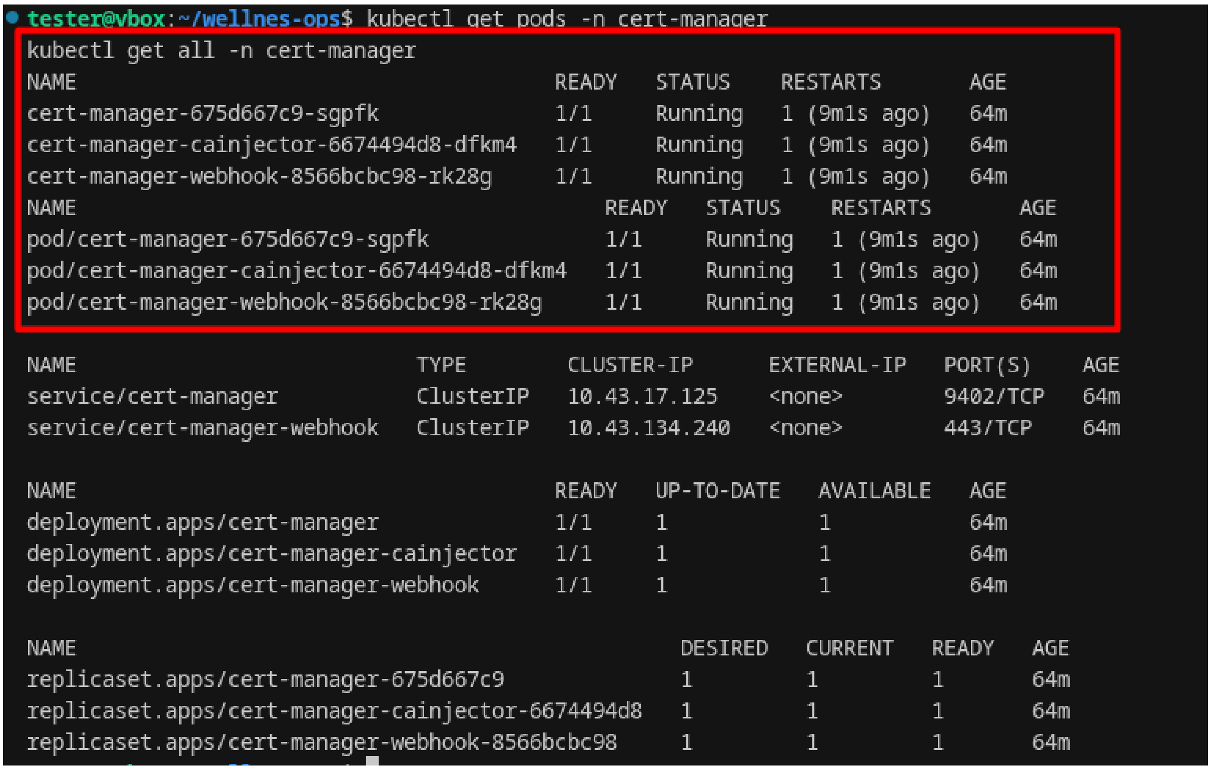
```
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --set crds.enabled=true
```

Verify that Cert-Manager pods are running:

```
kubectl get pods -n cert-manager
```

List all resources in the Cert-Manager namespace:

```
kubectl get all -n cert-manager
```



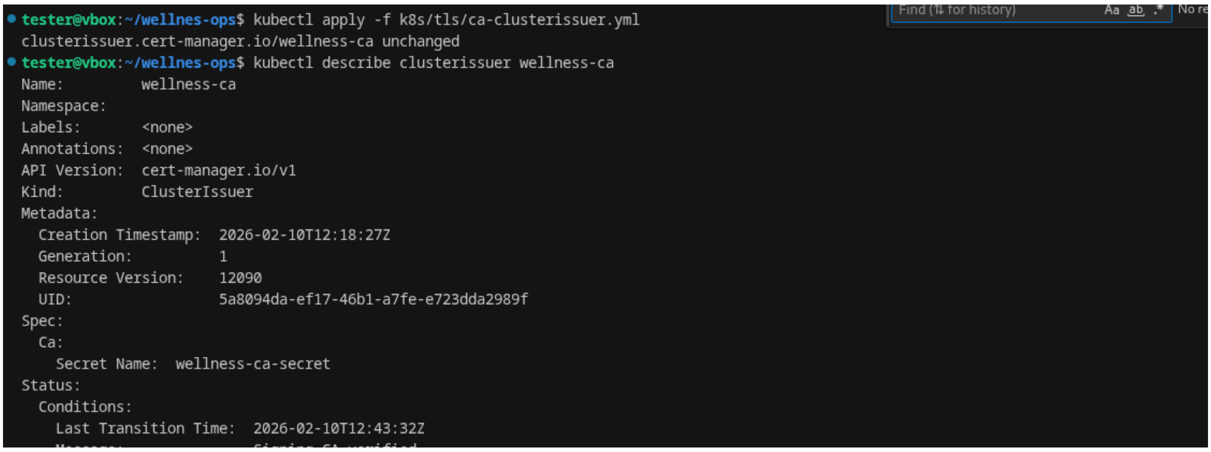
```
tester@vbox:~/wellnes-ops$ kubectl get pods -n cert-manager
kubectl get all -n cert-manager
NAME                                      READY   STATUS    RESTARTS      AGE
cert-manager-675d667c9-sgpfk              1/1     Running   1 (9m1s ago)  64m
cert-manager-cainjector-6674494d8-dfkm4   1/1     Running   1 (9m1s ago)  64m
cert-manager-webhook-8566bcbc98-rk28g     1/1     Running   1 (9m1s ago)  64m
NAME                                          READY   STATUS    RESTARTS      AGE
pod/cert-manager-675d667c9-sgpfk              1/1     Running   1 (9m1s ago)  64m
pod/cert-manager-cainjector-6674494d8-dfkm4   1/1     Running   1 (9m1s ago)  64m
pod/cert-manager-webhook-8566bcbc98-rk28g     1/1     Running   1 (9m1s ago)  64m

NAME                           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/cert-manager           ClusterIP   10.43.17.125    <none>        9402/TCP   64m
service/cert-manager-webhook   ClusterIP   10.43.134.240   <none>        443/TCP    64m

NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cert-manager              1/1     1            1           64m
deployment.apps/cert-manager-cainjector   1/1     1            1           64m
deployment.apps/cert-manager-webhook      1/1     1            1           64m

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/cert-manager-675d667c9              1         1         1       64m
replicaset.apps/cert-manager-cainjector-6674494d8   1         1         1       64m
replicaset.apps/cert-manager-webhook-8566bcbc98     1         1         1       64m
```

Apply TLS Manifests (In Order)

Apply the TLS configuration manifests located in `/k8s/tls/`, one by one and in the correct order. This sequencing is important to ensure proper initialization of Issuers, Certificates, and all TLS-related resources required for HTTPS

kubectl apply -f k8s/tls/ca-clusterissuer.yml

kubectl describe clusterissuer wellness-ca



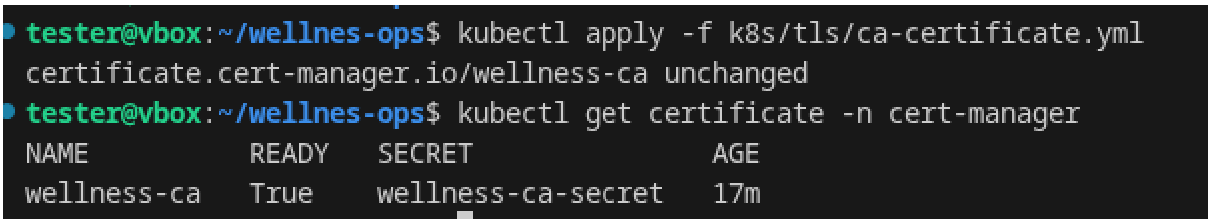
```
tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/tls/ca-clusterissuer.yml
clusterissuer.cert-manager.io/wellness-ca unchanged
tester@vbox:~/wellnes-ops$ kubectl describe clusterissuer wellness-ca
Name:         wellness-ca
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  cert-manager.io/v1
Kind:         ClusterIssuer
Metadata:
  Creation Timestamp:  2026-02-10T12:18:27Z
  Generation:          1
  Resource Version:    12090
  UID:                 5a8094da-ef17-46b1-a7fe-e723dda2989f
Spec:
  Ca:
    Secret Name:  wellness-ca-secret
Status:
  Conditions:
    Last Transition Time:  2026-02-10T12:43:32Z
    Message:               Signing CA verified
```

Apply the CA-signed Certificate manifest:

kubectl apply -f k8s/tls/ca-certificate.yml

kubectl get certificate -n cert-manager

kubectl get certificate -n cert-manager



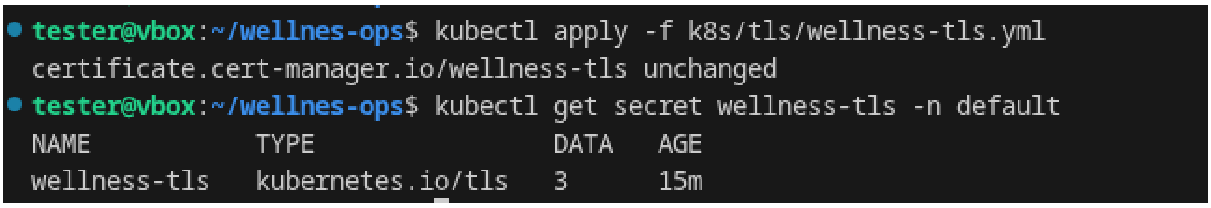
```
tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/tls/ca-certificate.yml
certificate.cert-manager.io/wellness-ca unchanged
tester@vbox:~/wellnes-ops$ kubectl get certificate -n cert-manager
NAME          READY   SECRET               AGE
wellness-ca   True    wellness-ca-secret   17m
```

Apply the TLS certificate manifest:

kubectl apply -f k8s/tls/wellness-tls.yml

Verify that the TLS secret has been created in the `default` namespace:

kubectl get secret wellness-tls -n default



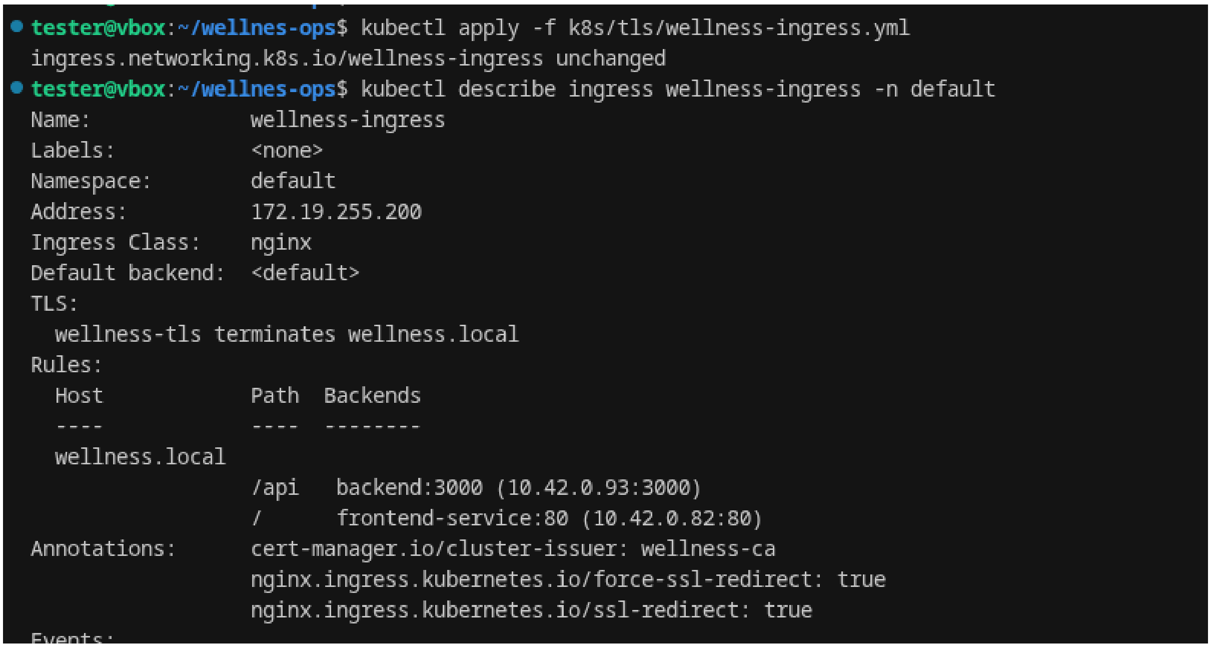
```
● tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/tls/wellness-tls.yml
  certificate.cert-manager.io/wellness-tls unchanged
● tester@vbox:~/wellnes-ops$ kubectl get secret wellness-tls -n default
  NAME           TYPE                DATA   AGE
  wellness-tls   kubernetes.io/tls   3      15m
```

Apply the Ingress manifest for the application:

kubectl apply -f k8s/tls/wellness-ingress.yml

Verify that the Ingress resource has been created and is configured correctly:

kubectl describe ingress wellness-ingress -n default



```
● tester@vbox:~/wellnes-ops$ kubectl apply -f k8s/tls/wellness-ingress.yml
  ingress.networking.k8s.io/wellness-ingress unchanged
● tester@vbox:~/wellnes-ops$ kubectl describe ingress wellness-ingress -n default
  Name:             wellness-ingress
  Labels:           <none>
  Namespace:        default
  Address:          172.19.255.200
  Ingress Class:    nginx
  Default backend:  <default>
  TLS:
    wellness-tls terminates wellness.local
  Rules:
    Host            Path  Backends
    ----            ----  --------
    wellness.local
                    /api   backend:3000 (10.42.0.93:3000)
                    /      frontend-service:80 (10.42.0.82:80)
  Annotations:      cert-manager.io/cluster-issuer: wellness-ca
                    nginx.ingress.kubernetes.io/force-ssl-redirect: true
                    nginx.ingress.kubernetes.io/ssl-redirect: true
  Events:
```

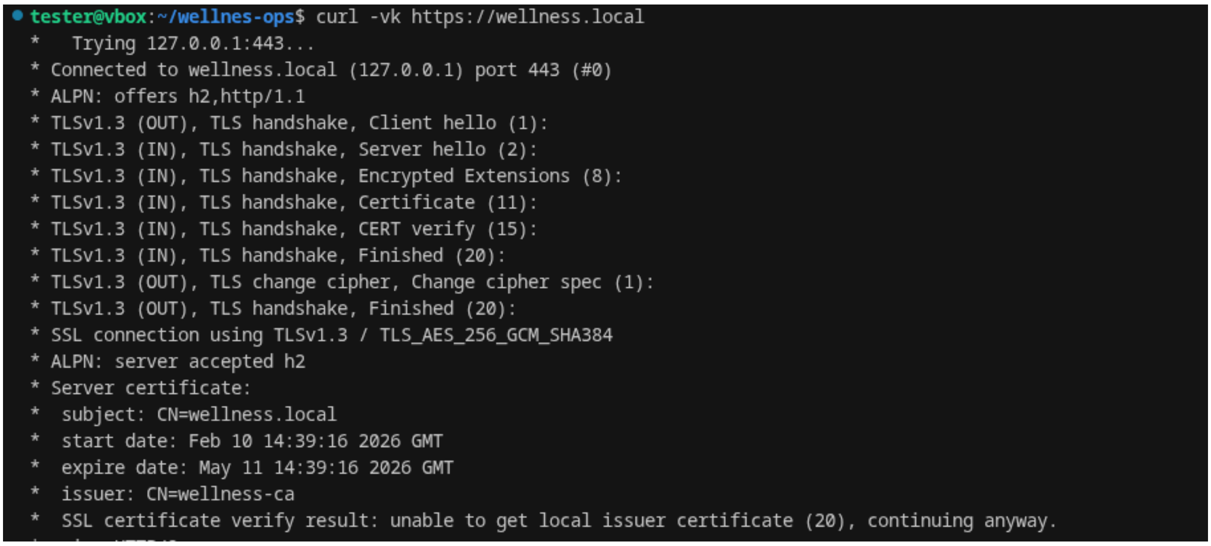
Verify Application Access

Confirm that the application is accessible.

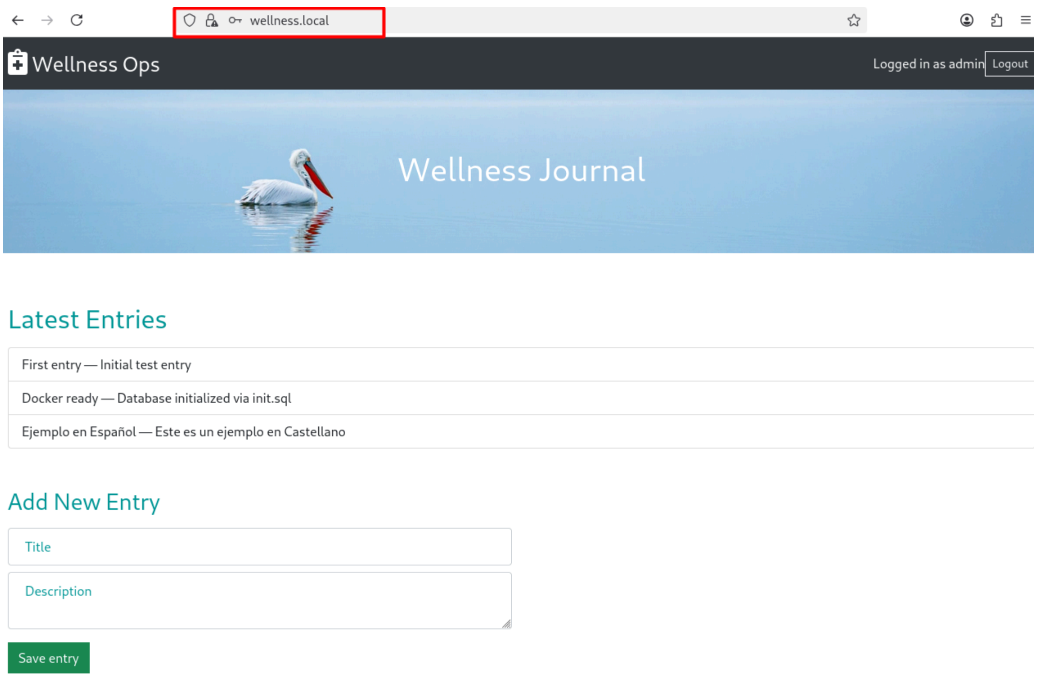
Once all components are deployed and all TLS resources are applied, validate that the application is reachable through its HTTPS endpoint.

Typical checks include:

- Accessing the application URL in a browser
- Confirming that the TLS certificate is valid
- Ensuring that the Ingress routes traffic correctly
- Verifying that the backend responds without errors



```
tester@vbox:~/wellnes-ops$ curl -vk https://wellness.local
*   Trying 127.0.0.1:443...
* Connected to wellness.local (127.0.0.1) port 443 (#0)
* ALPN: offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN: server accepted h2
* Server certificate:
*  subject: CN=wellness.local
*  start date: Feb 10 14:39:16 2026 GMT
*  expire date: May 11 14:39:16 2026 GMT
*  issuer: CN=wellness-ca
*  SSL certificate verify result: unable to get local issuer certificate (20), continuing anyway.
```



wellness.local

Wellness Ops

Logged in as admin

Wellness Journal

Latest Entries

First entry — Initial test entry

Docker ready — Database initialized via init.sql

Ejemplo en Español — Este es un ejemplo en Castellano

Add New Entry