Memgraph in K8s

Andi Skrgat

Content of this deck

1. Motivation
2. K8s concepts
3. Helm charts

01

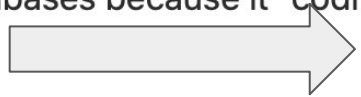
Motivation

👉🏻 Purpose/value statement - *Explain why the company will invest resources in this effort*

1. A poll under CTOCraft showed that ~40% of new software projects are using k8s
2. Kubernetes provides the highest level of abstraction to manage applications (including databases)
3. Database operators (sys admin and dev ops folks) prefer k8s because it gives the ability for database vendors to directly provide code to deal with all operational concerns, it's the best way to deploy databases because it "codifies operational knowledge" (NOTE: that's the operator part)p

👉🏻 Purpose/value statement - *Explain why the company will invest resources in this effort*

1. A poll under CTOCraft showed that ~40% of new software projects are using k8s
2. Kubernetes provides the highest level of abstraction to manage applications (including databases)
3. Database operators (sys admin and dev ops folks) prefer k8s because it gives the ability for database vendors to directly provide code to deal with all operational concerns, it's the best way to deploy databases because it "codifies operational knowledge" (NOTE: that's the operator part)p



Not just operator!

02

K8s

K8s

- Borg’s successor
- **Container** orchestration platform that automates deployment, scaling and management of containerized applications
- Containers are grouped together into **pods**
- Hosted by Cloud Native Computing Foundation (CNCF)

How to create “something” in K8s?

- Imperative approach: e.g. `kubectl run nginx-pod --image=nginx:latest`
  - Fast for testing and debugging ✅
  - No YAML required ✅
  - Hard to track changes 🛑
  - Not reproducible 🛑
- Declarative approach
  - Version-controlled and reproducible ✅
  - Supports GitOps ✅
  - Requires YAML knowledge 🛑

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    env: demo
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80  # Note: "P" must be uppercase!
```

Pods & containers

- Pod is a collection of containers with shared storage and network resources
  - Shared storage means shared volumes
  - Each container has its own filesystem
  - Network namespace is completely shared, same IP address, same port space
  - Process namespace of each container is private (but can be shared somehow)
  - IPC namespace → shared memory segments, semaphores
- There are also init-containers which are run at pod startup
- Ephemeral containers useful for debugging because of shared-process namespace feature

K8s storage

- **Persistent Volume Claims (PVC)** ⇒ a user's request for storage with specific size and access mode.
- **Persistent Volume (PV)** ⇒ a piece of actual storage provisioned in the cluster (EBS volume, local disk, NFS...).
  - 1v1 mapped to PVC
  - Durable storage that outlives any single pod

- **Volumes** ⇒ for containers in a pod to share to access and share data via the filesystem
  - Solve 2 problems:
    1. On-disk files in a container are ephemeral
    2. Sharing data between containers

K8s storage

- Volume mounts ⇒ configuration inside the container
- “Take pod’s volume and make it accessible at this path inside the container”

	Inline Volume	PV/PVC
Lifecycle	Tied to Pod	Independent of Pod
Survives Pod restart	No (most types)	Yes
Provisioning	Inline in Pod spec	Separate objects, can be dynamic
Use case	Temp files, configs, secrets	Databases, persistent state
Shared across Pods	Limited	Yes (with ReadWriteMany)

StatefulSet

- You need it when:
  - Stable, unique network identifiers
  - Stable, persistent storage
  - Ordered, graceful deployment and scaling
  - Ordered, automated rolling updates

- A collection of pods
- In Memgraph, we have one StatefulSet for each data instance and for each coordinator
- In the future, try to set-up replicas for a StatefulSet (one StatefulSet for data instances and one StatefulSet for coordinators)
- Running as non-root memgraph user

Services

- Each pod gets its own unique IP address (within the cluster)
- All pods can communicate with each other regardless of the node on which they are deployed
- Every time the pod restarts, it gets a new IP address ⇒ SERVICE
- Service provides you a stable IP address/hostname which persists after pod restart
- ClusterIP
- To enable client-access from the outside of the cluster, several options (later)

NodePort

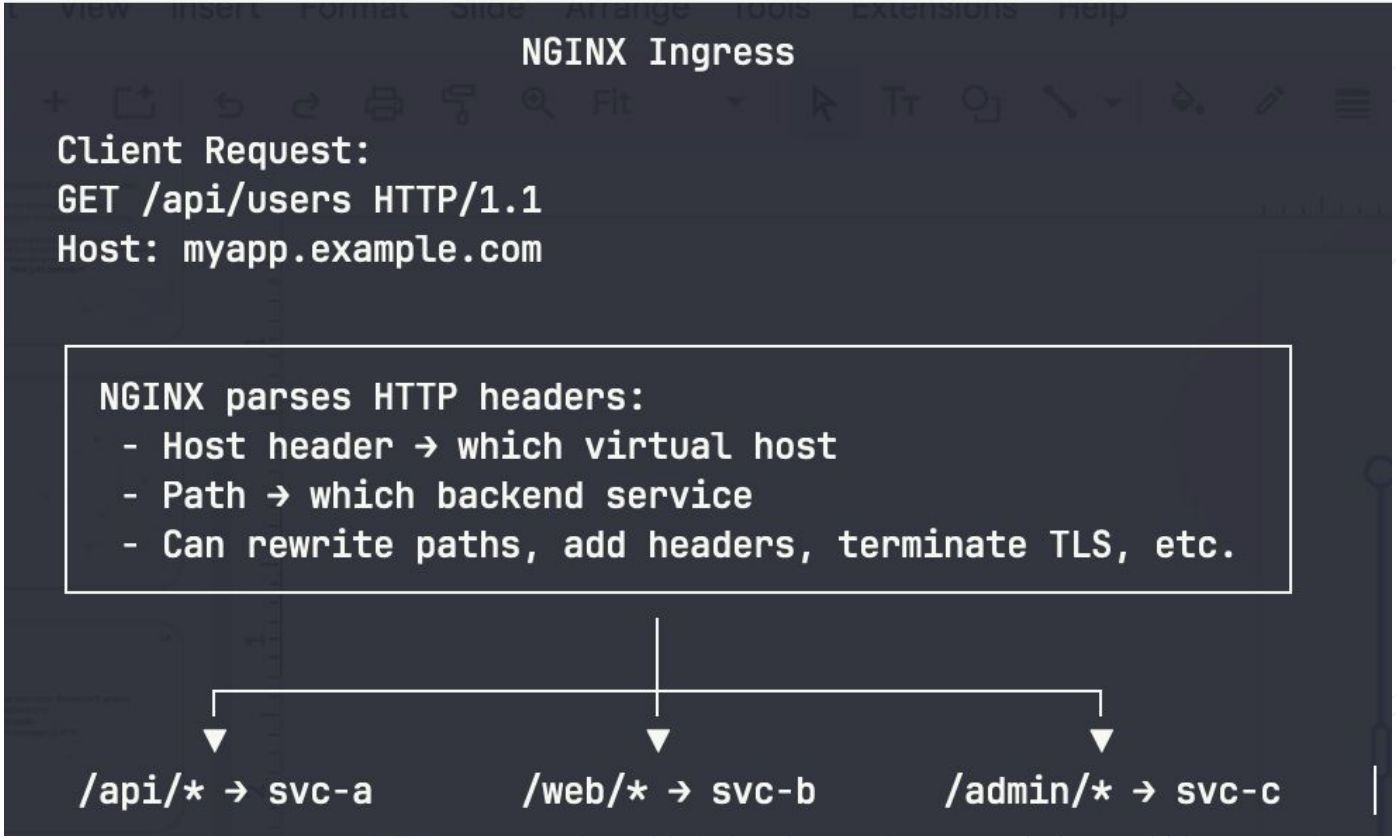
- Exposes the Service on each Node's IP at a static port (the NodePort). To make the node port available, Kubernetes sets up a cluster IP address, the same as if you had requested a Service of type: ClusterIP.
- Security concerns but simplest without costs

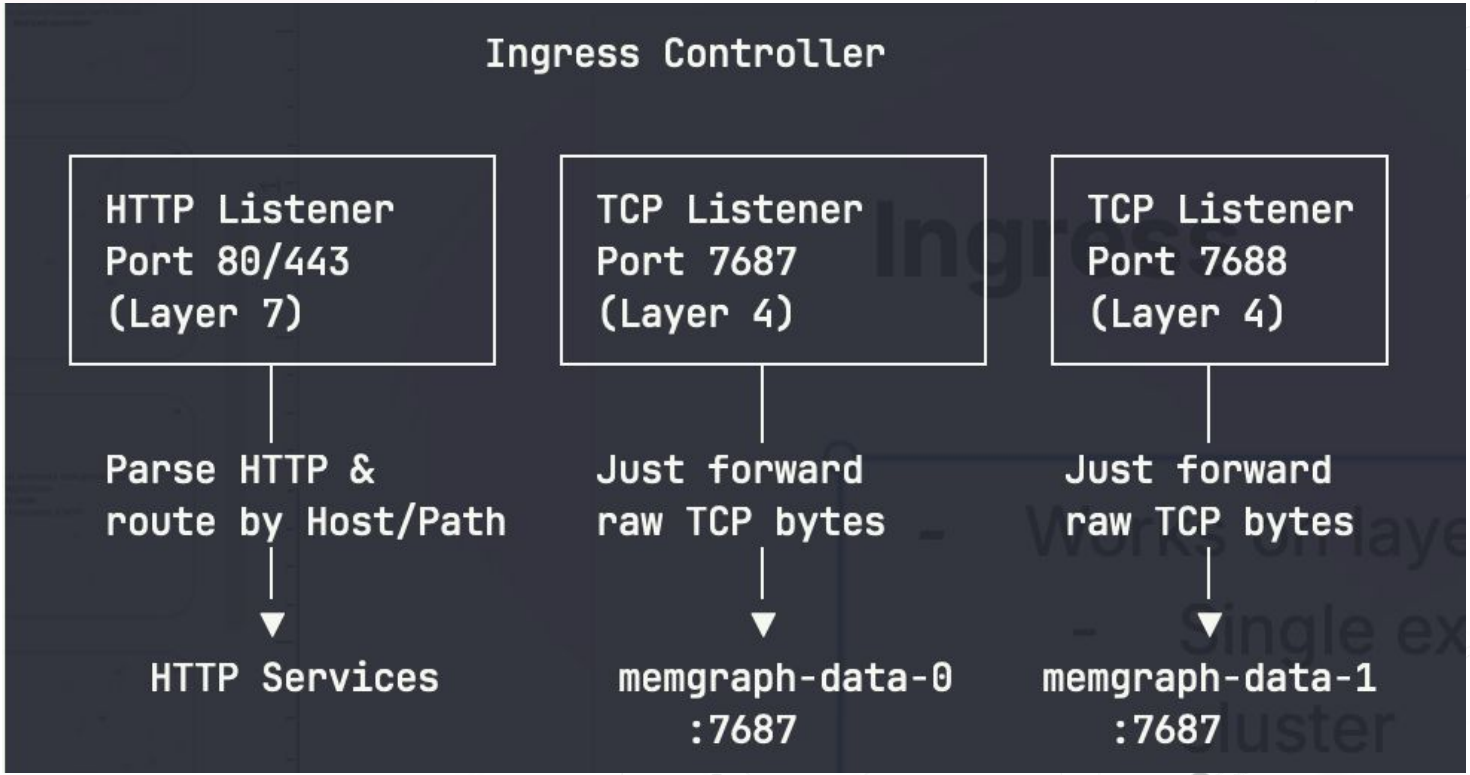
Load Balancer

- Exposes the Service externally using an external load balancer. Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider
- Load balancer gets its own external IP
- CommonLoadBalancer (our object) ⇒ one LoadBalancer for all coordinators + one coordinator for each data instance

Ingress

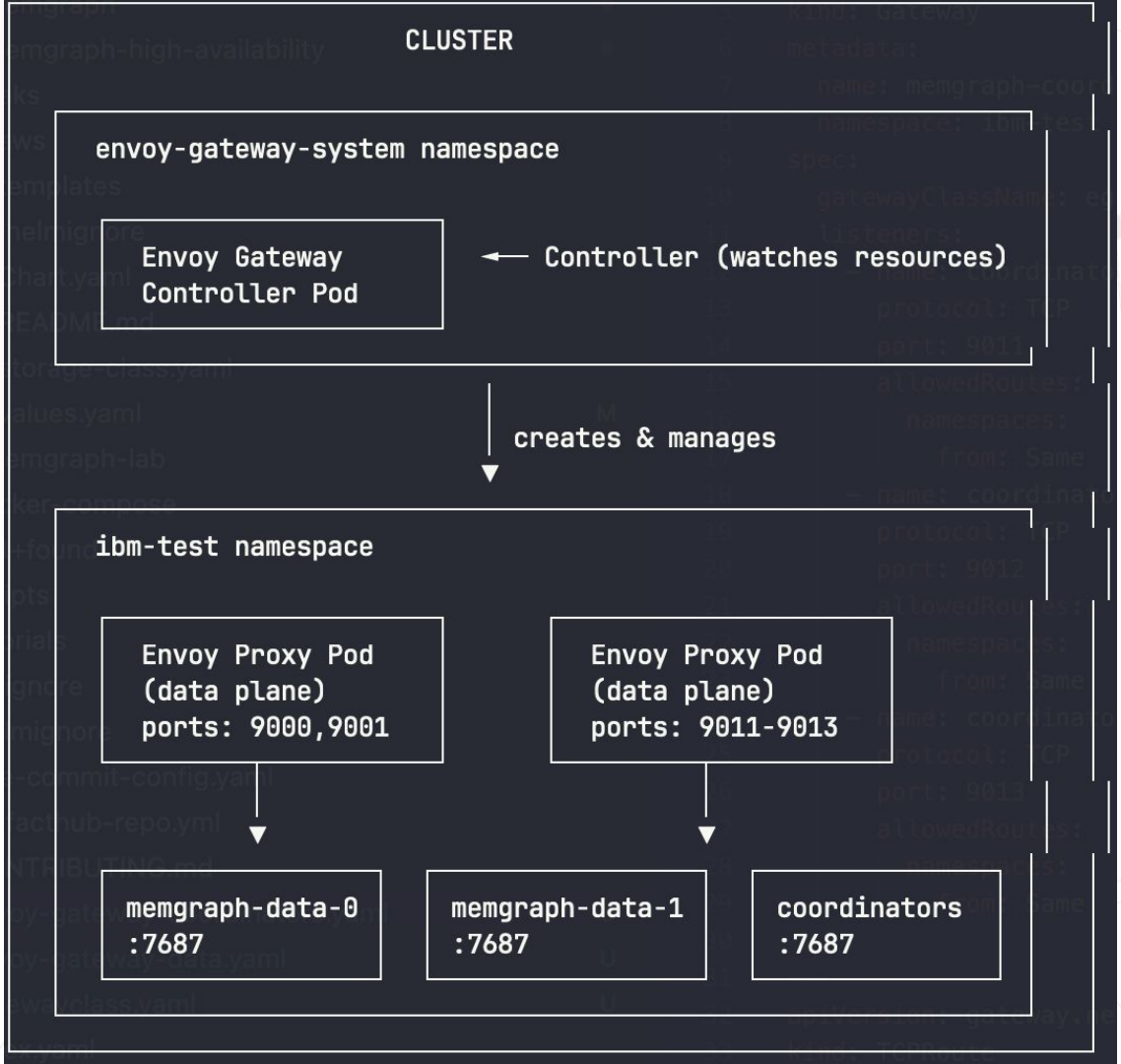
- Works on layer 7 (HTTP/HTTPS)
  - Single external IP and then route to specific service within the K8s cluster
  - Understands web concepts (URIs, hostnames, paths...)
- Bolt works on Layer 4 (TCP) which means we cannot use Ingress directly
- We combined it with nginx to forward TCP-level bytes


NGINX Ingress
Client Request:
GET /api/users HTTP/1.1
Host: myapp.example.com
NGINX parses HTTP headers:
- Host header → which virtual host
- Path → which backend service
- Can rewrite paths, add headers, terminate TLS, etc.
/api/* → svc-a
/web/* → svc-b
/admin/* → svc-c


Ingress Controller
HTTP Listener
Port 80/443
(Layer 7)
TCP Listener
Port 7687
(Layer 4)
TCP Listener
Port 7688
(Layer 4)
Parse HTTP &
route by Host/Path
Just forward
raw TCP bytes
Just forward
raw TCP bytes
HTTP Services
memgraph-data-0
:7687
memgraph-data-1
:7687

Gateway

- modern solution. One hostname + port-based access to internal services within the cluster. No support out of the box, but example on docs.


CLUSTER
envoy-gateway-system namespace
Envoy Gateway Controller Pod
Controller (watches resources)
creates & manages
ibm-test namespace
Envoy Proxy Pod (data plane) ports: 9000,9001
Envoy Proxy Pod (data plane) ports: 9011-9013
memgraph-data-0 :7687
memgraph-data-1 :7687
coordinators :7687

GatewayClass

- Choose which controller (Envoy, NGINX, Istio) handles your gateways

gatewayclass.yaml > { } spec

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: eg
spec:
  controllerName: gateway.envoyproxy.io/gatewayclass-controller
```

Gateway

- The actual load balancer/proxy that listens on ports
- Controller will expose port 9000 externally

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: memgraph-data-gateway
spec:
  gatewayClassName: eg          # Use the Envoy controller
  listeners:
    - name: data-0
      protocol: TCP             # Listen for TCP traffic
      port: 9000                # On this external port
```

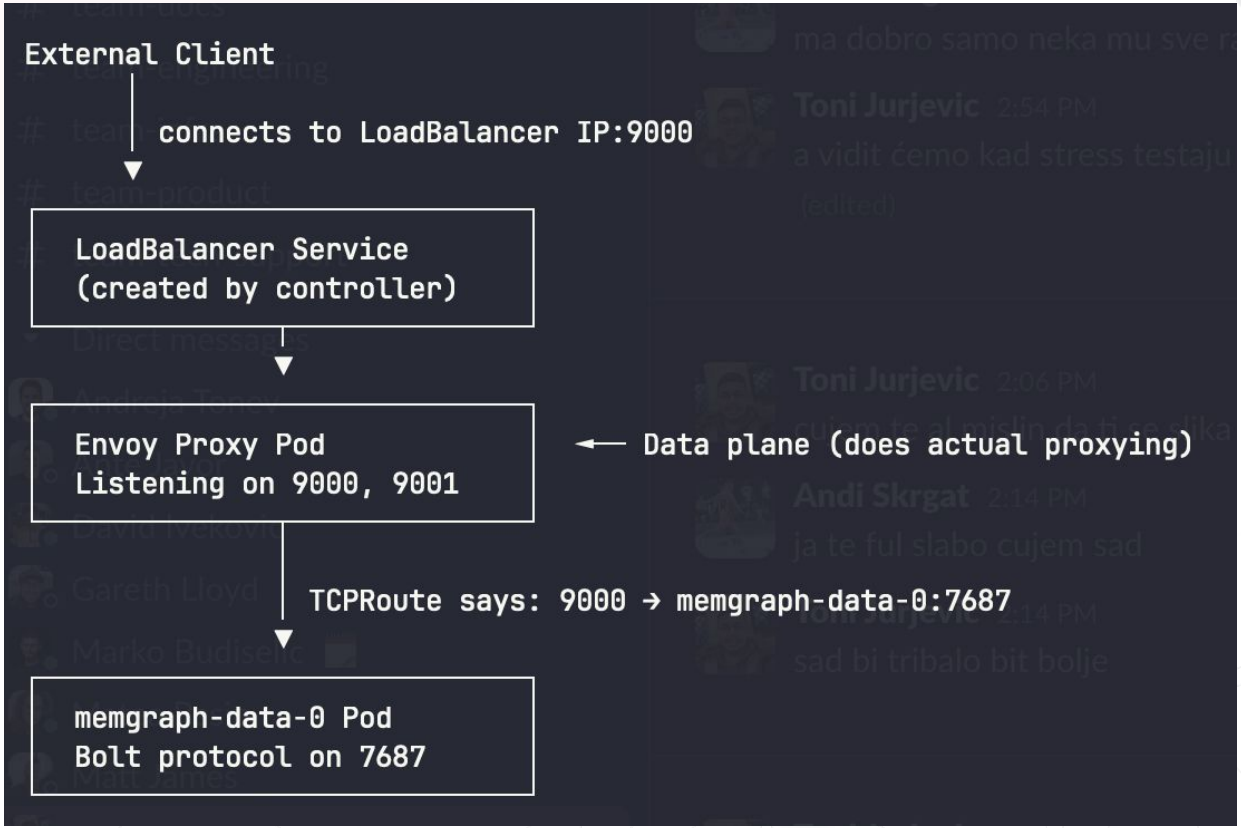
TCPRoute

- Routing rules → where to send the traffic that arrives on a specific listener

```yaml
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: data-0-route
spec:
  parentRefs:
    - name: memgraph-data-gateway    # Attach to this Gateway
      sectionName: data-0            # Specifically the data-0 listener
  rules:
    - backendRefs:
        - name: memgraph-data-0      # Forward to this Service
          port: 7687
```

Connection flow

-



Headless services

- bypasses the K8s service proxy and returns the pod IPs directly via DNS
- it allows application to control to which pod it should get connected to
- gives each pod a stable DNS name

```bash
# Regular service
nslookup my-service → 10.96.0.1 (ClusterIP)

# Headless service
nslookup my-service → 10.244.0.5, 10.244.0.6, 10.244.0.7 (pod IPs)

# Headless service gives each pod a stable DNS name:
memgraph-0.memgraph-svc.default.svc.cluster.local
memgraph-1.memgraph-svc.default.svc.cluster.local
memgraph-2.memgraph-svc.default.svc.cluster.local
```

03

MG Helm charts

Helm charts

- The package manager for K8s
- Define, install and upgrade K8s applications
- Values.yaml ⇒ a set of configuration values you give to your K8s application
- Template files ⇒ app logic

Install chart

commands.txt

```
# Standalone

helm install memgraph-db helm-charts/charts/memgraph -f values.yaml
helm install memgraph-db memgraph/memgraph -f values.yaml // Official chart

# HA
helm install memgraph-db helm-charts/charts/memgraph-high-availability -f values.yaml
helm install memgraph-db memgraph/memgraph-high-availability -f values.yaml // Official chart
```

Install charts locally

```
minikube start --nodes=5 --driver=docker
```

Install HA charts locally

1 Enable csi-hostpath-driver

```
minikube addons disable storage-provisioner
minikube addons disable default-storageclass
minikube addons enable volumesnapshots
minikube addons enable csi-hostpath-driver
```

Install HA charts locally

2 Create a StorageClass (save as sc.yaml)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-hostpath-delayed
provisioner: hostpath.csi.k8s.io
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Delete
```

Install HA charts locally

3 Apply the StorageClass

```
kubectl apply -f sc.yaml
```

4 Configure the Helm chart

In your `values.yaml`, set:

```
storage:
  libStorageClassName: csi-hostpath-delayed
```

07

Tools

Tools + Demo

- Minikube for local testing
- Kubectl as command-line tool
- https://k9scli.io/
- Azure AKS
- Docker

Thank you for your time!

MEM GRAPH

www.memgraph.com