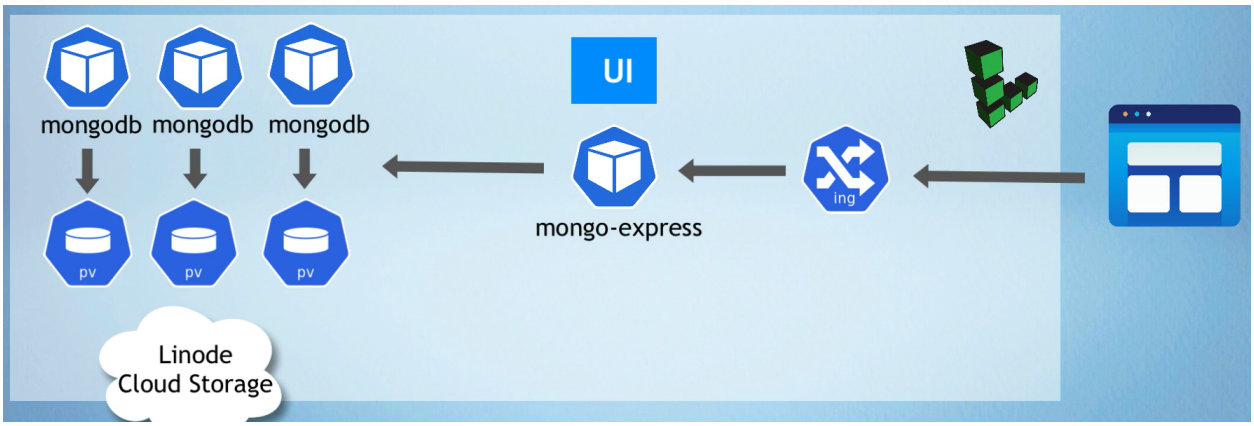
Demo Project: Install a stateful (MongoDB) on Kubernetes using Helm

This guide demonstrates how to deploy a stateful MongoDB instance on a Kubernetes cluster using Helm, configure data persistence, and expose a UI client using Nginx ingress.

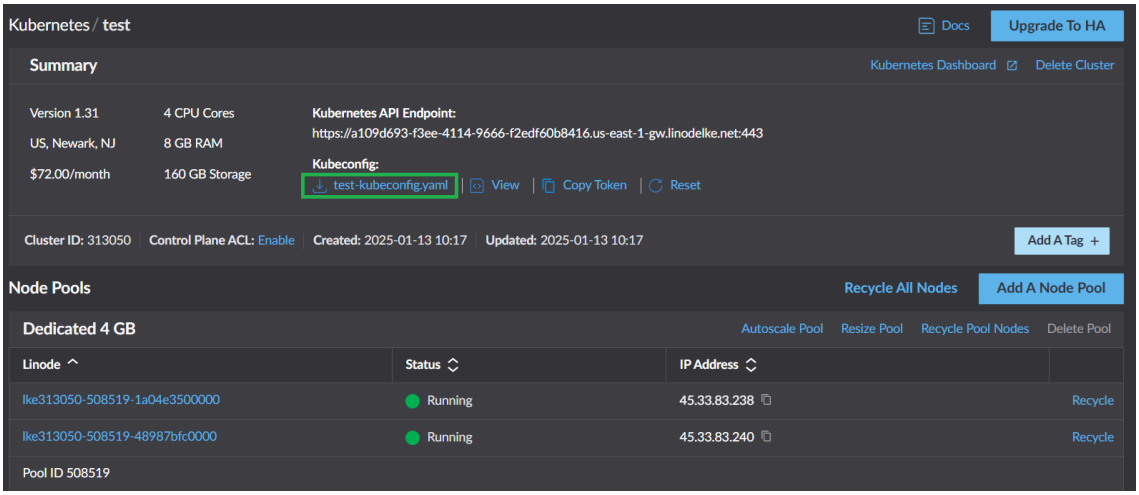


Step 1: Create a managed K8s cluster with Linode Kubernetes Engine (LKE)

1. **Set Up a Kubernetes Cluster**:

- Navigate to **Create → Kubernetes** on the Linode dashboard.
- Fill in the following details:
  - **Cluster Label**: `Test`
  - **Region**: Select a region closest to you.
  - **Kubernetes Version**: Choose the latest available version.
- Add Node Pools:
  - Add at least **2 nodes** with `Dedicated 4GB` size.
- Click **Create Cluster**.

2. **Download the Kubeconfig File**:
   - Once the cluster is created, download the `test-kubeconfig.yaml` file. This file contains credentials and certificates needed to access the Kubernetes cluster remotely.



3. **Secure Kubeconfig File**:
   - Place the `test-kubeconfig.yaml` in your working directory and restrict access:
     `chmod 400 test-kubeconfig.yaml`
   - Export the file as an environment variable:

     `export KUBECONFIG=test-kubeconfig.yaml`

- **How it works:** The above command is used to configure your local Kubernetes CLI (`kubectl`) to interact with the **Kubernetes cluster** hosted on Linode's Kubernetes Engine (LKE).

  The `test-kubeconfig.yaml` file contains the credentials, certificates, and API server endpoint information required for secure communication with the remote Kubernetes cluster. By exporting it as an environment variable, the `kubectl` CLI knows where to find the cluster configuration to manage Kubernetes resources such as nodes, pods, services, etc., within the Linode Kubernetes Engine.

4. **Verify the Cluster**:
   - Confirm that the nodes are running: `kubectl get node`

Step 2: Deploy replicated MongoDB service in LKE cluster using a Helm chart & Configure data persistence for MongoDB with Linode's cloud storage

1. **Install Helm**:
   - Follow the Helm installation guide (https://helm.sh/docs/intro/install/) for your operating system.
2. **Add the Bitnami Repository**:
   - Add the Helm repository for Bitnami charts:

     `helm repo add bitnami https://charts.bitnami.com/bitnami`
3. **Search for MongoDB Charts**:
   - List all charts in the Bitnami repository: `helm search repo bitnami`
   - Find MongoDB-specific charts: `helm search repo bitnami/mongodb`

4. **Override Default Values**:
    - The chart provide defaults parameters and values seen here: https://github.com/bitnami/charts/tree/main/bitnami/mongodb
    - To override values for example these:

charts / bitnami / mongodb /

MongoDB(®) parameters

Name	Description	Value
`image.registry`	MongoDB(®) image registry	`REGISTRY_NAME`
`image.repository`	MongoDB(®) image registry	`REPOSITORY_NAME/mongodb`
`image.digest`	MongoDB(®) image digest in the way sha256:aa.... Please note this parameter, if set, will override the tag	`""`
`image.pullPolicy`	MongoDB(®) image pull policy	`IfNotPresent`
`image.pullSecrets`	Specify docker-registry secret names as an array	`[]`
`image.debug`	Set to true if you would like to see extra information on logs	`false`
`schedulerName`	Name of the scheduler (other than default) to dispatch pods	`""`
`architecture`	MongoDB(®) architecture (`standalone` or `replicaset`)	`standalone`
`useStatefulSet`	Set to true to use a StatefulSet instead of a Deployment (only when `architecture=standalone`)	`false`
`auth.enabled`	Enable authentication	`true`
`auth.rootUser`	MongoDB(®) root user	`root`
`auth.rootPassword`	MongoDB(®) root password	`""`

Create a `helm-mongodb.yaml` values file to configure MongoDB and data persistence. Example configuration:

```
architecture: replicaset
replicaCount: 3
persistence:
  storageClass: "linode-block-storage"
auth:
  rootPassword: secret-root-pwd
```

5. **Deploy MongoDB**:
   - Use the custom values file to deploy MongoDB:

   `helm install [our name] --values [values file name] [chart name]`

   Example: `helm install mongodb --values helm-mongodb.yaml bitnami/mongodb`
6. **Verify Deployment**:
   - Check that the MongoDB pods are running: `kubectl get pod`

Step 3: Deploy UI client Mongo Express for MongoDB

1. **Prepare the Configuration**:
   - Since we need only 1 pod and 1 service for the mongo-express no need to create a helm chart, create a `helm-mongo-express.yaml` file for Mongo Express.
2. **Deploy Mongo Express**:
   - Apply the configuration:

     `kubectl apply -f helm-mongo-express.yaml`
3. **Verify Deployment**:
   - Check that the Mongo Express pod is running: `kubectl get pods`

Step 4: Deploy and configure nginx ingress to access the UI application from browser

1. **Deploy nginx-ingress controller using Helm Chart:**
    - Add the Nginx-Ingress Helm repository:

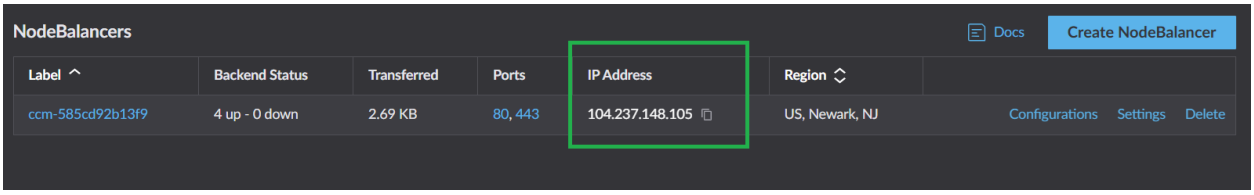
      `helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx`
    - Install the Nginx-Ingress Helm chart:

      `helm install [our name] [chart name] --set controller.publishService.enabled=true`

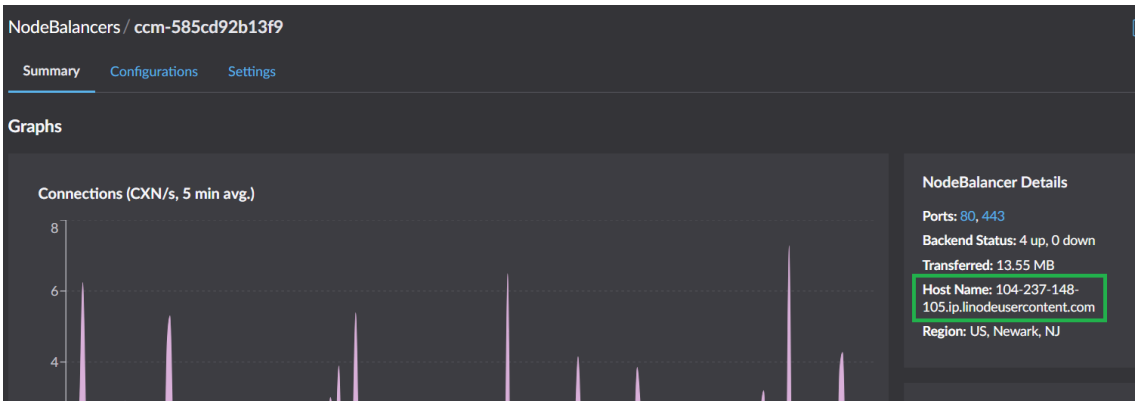
      Example:

      `helm install nginx-ingress ingress-nginx/ingress-nginx --set controller.publishService.enabled=true`

      Note: `--set controller.publishService.enabled=true` is to make sure that we are automatically allocated a public IP for our ingress address to use with nginx.
2. **Verify the Deployment:**
    - Confirm that the Nginx-Ingress pods are running: `kubectl get pods`
    - Obtain the public IP from Linode's **NodeBalancers** section in the console.

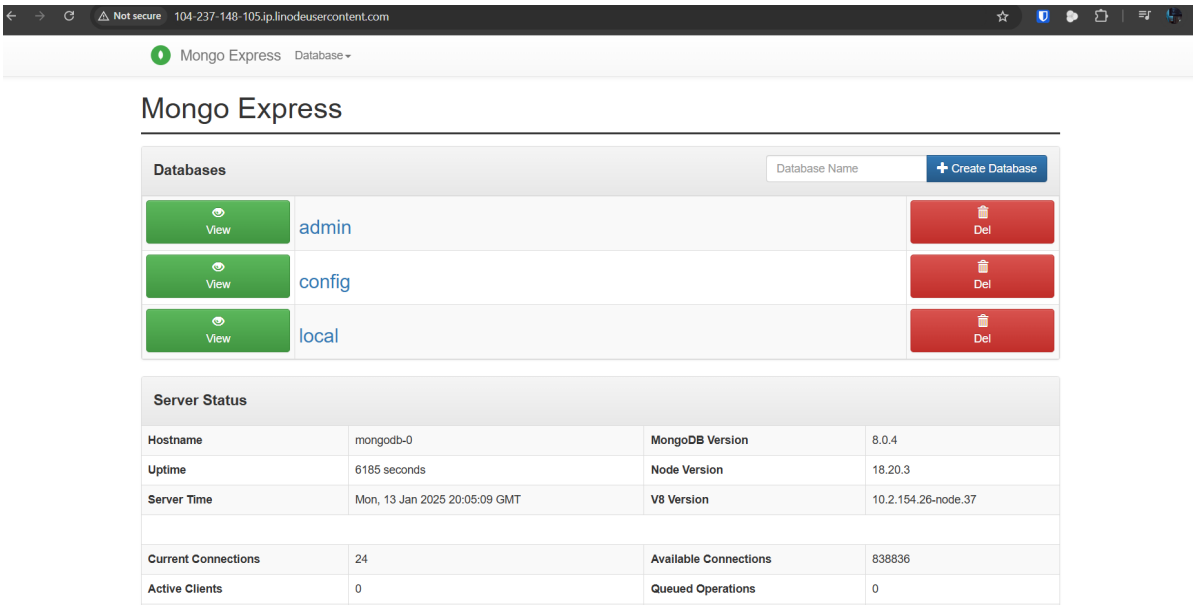


3. **Create Ingress Rule:**
    - Take note of the host name from Linodes NodeBalancers to be able to add in the ingress rule.



- Create a `helm-ingress-yaml` file.
- Apply the ingress configuration: `kubectl apply -f helm-ingress.yaml`

4. **Access the Application**:
   - Open the public IP in a browser to access Mongo Express:



Step 5: Test Data Persistence

1. **Scale Down MongoDB**:
   - Scale down the MongoDB StatefulSet to simulate a restart: `kubectl scale --replicas=0 statefulset/mongodb`

2. **Scale MongoDB Back Up**:
    - Scale up the MongoDB StatefulSet: `kubectl scale --replicas=3 statefulset/mongodb`
3. **Verify Data Persistence**:
    - In Linode we can observe how the volumes get un-attached and then re-attached to the pods.
    - Check that the previously stored data is still available in Mongo Express.

Step 6: Clean Up Resources

1. **Uninstall MongoDB**:
    - Remove the MongoDB deployment: `helm uninstall mongodb`
    - **Note**: Persistent volumes must be manually deleted from the Linode console.
2. **Delete the Kubernetes Cluster**:
    - If you no longer need the cluster, delete it from the Linode console. Ensure to manually delete the **NodeBalancer** resource.

Step 7: Secure Your Workflow

GitHub Security Tip:

- Exclude the `test-kubeconfig.yaml` from version control by adding it to `.gitignore`

Why?

- This file contains sensitive credentials and certificates that can compromise your cluster if exposed.
