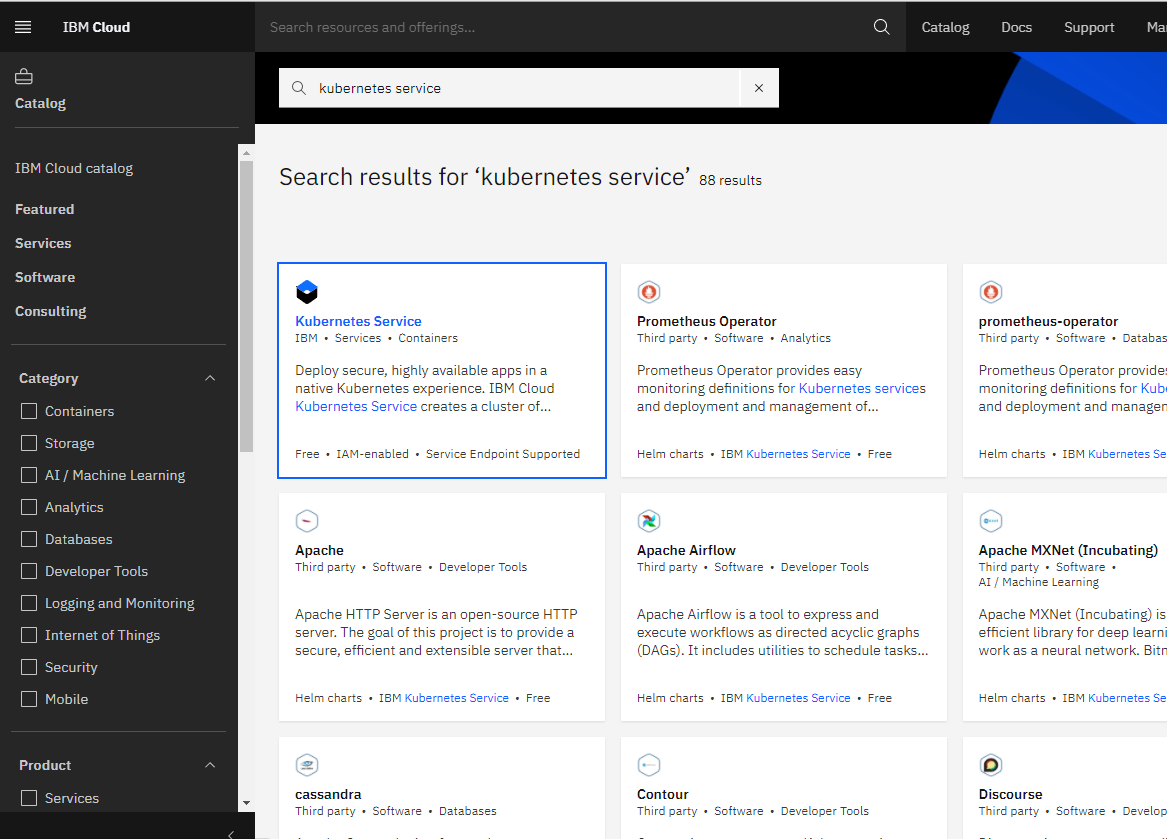
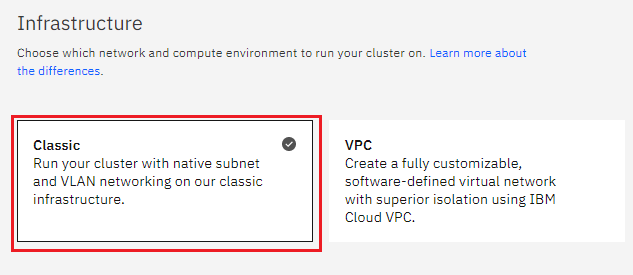
# Installing MongoDB on IBM Cloud

**Step 1 provision Kubernetes Cluster**

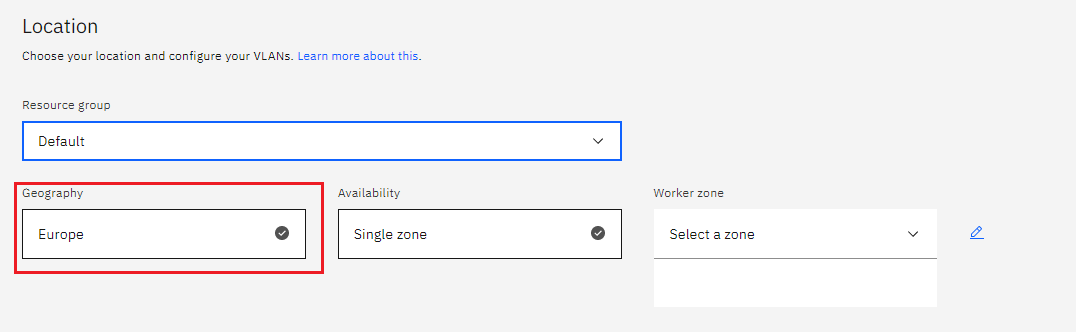
* Click the **Catalog** button on the top
* Select **Service** from the **Catalog**
* Search for **Kubernetes Service** and click on it



* You are now at the Kubernetes deployment page. You need to specify some details about the cluster
* Choose a plan **standard** or **free**. The free plan only has one worker node and no subnet, to provision a standard cluster, you will need to upgrade your account to Pay-As-You-Go
* To upgrade to a Pay-As-You-Go account, complete the following steps:
  + In the console, go to Manage > Account.
  + Select Account settings; and click Add credit card.
  + Enter your payment information, click Next, and submit your information
  + Choose **classic** or **VPC**, read the docs and choose the most suitable type for yourself



* Now choose your location settings,
* Choose **Geography** (continent)



* + Choose Single or Multizone, in single zone your data is only kept on datacentre, on the

other hand with Multizone it is distributed to multiple zones, thus safer in an unforeseen

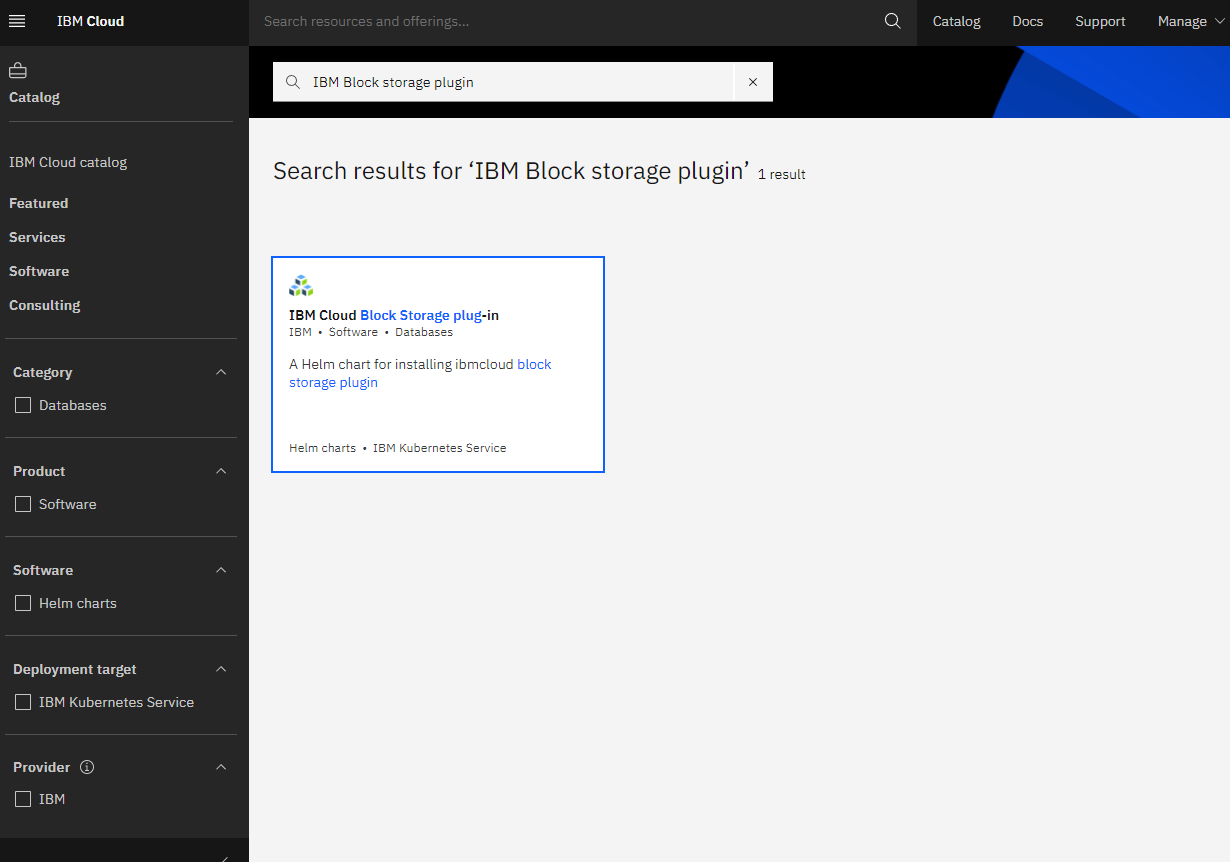
zone failure.

* If you wish to use Multizone, please set up your account with VRF.
* If at your current location selection, there is no available Virtual LAN, a new VLAN will be created for you.
* Choose a Worker node setup or use the preselected one, set Worker node amount per zone
* Choose **Master Service Endpoint** , In VRF-enabled accounts, you can choose private-only to make your master accessible on the private network or via VPN tunnel. Choose public-only to make your master publicly accessible. When you have a VRF-enabled account, your cluster is set up by default to use both private and public endpoints.  
  Give desired **tags** to your cluster, for more information visit tags
* Click **create**  
  • Wait for your cluster to be provisioned  
  • Your cluster is ready for usage

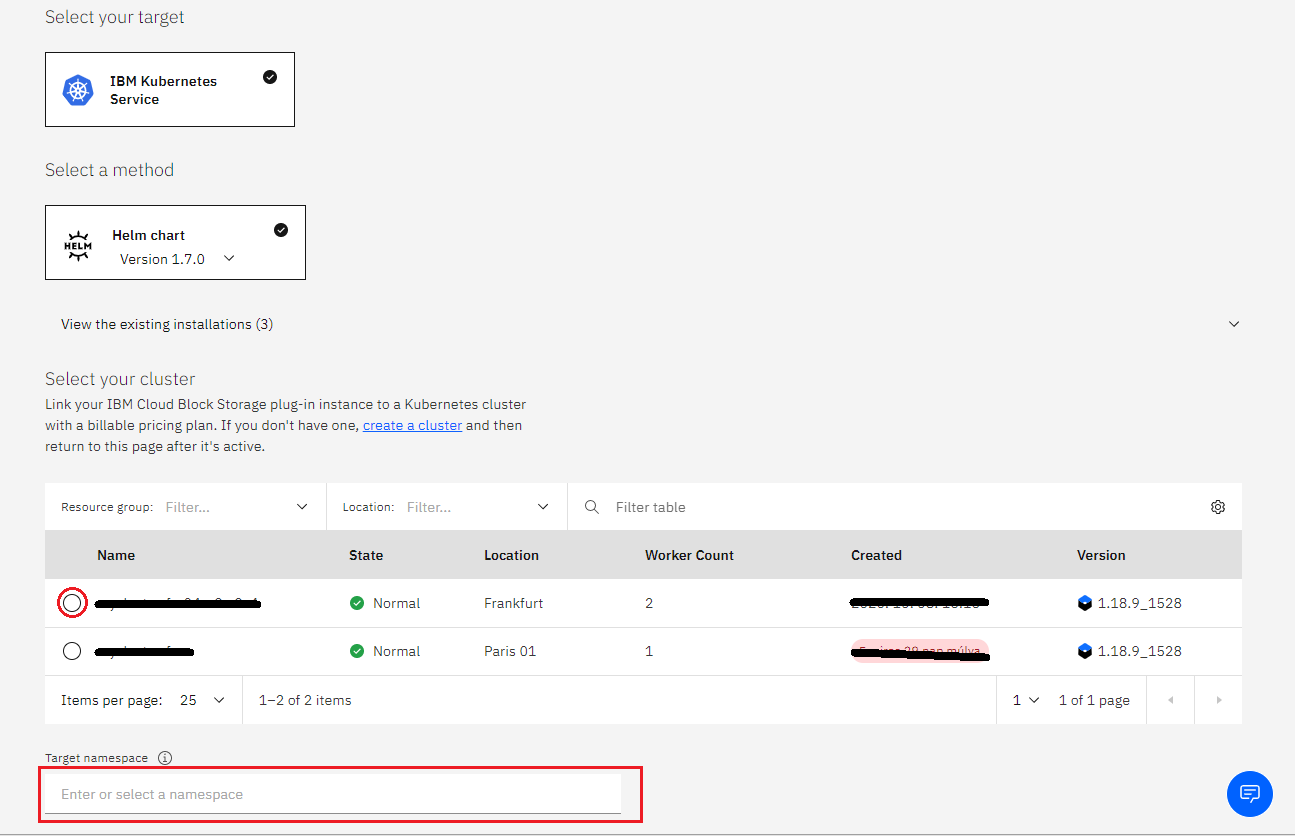
**Step 2 Deploy IBM Cloud Block Storage plug-in**

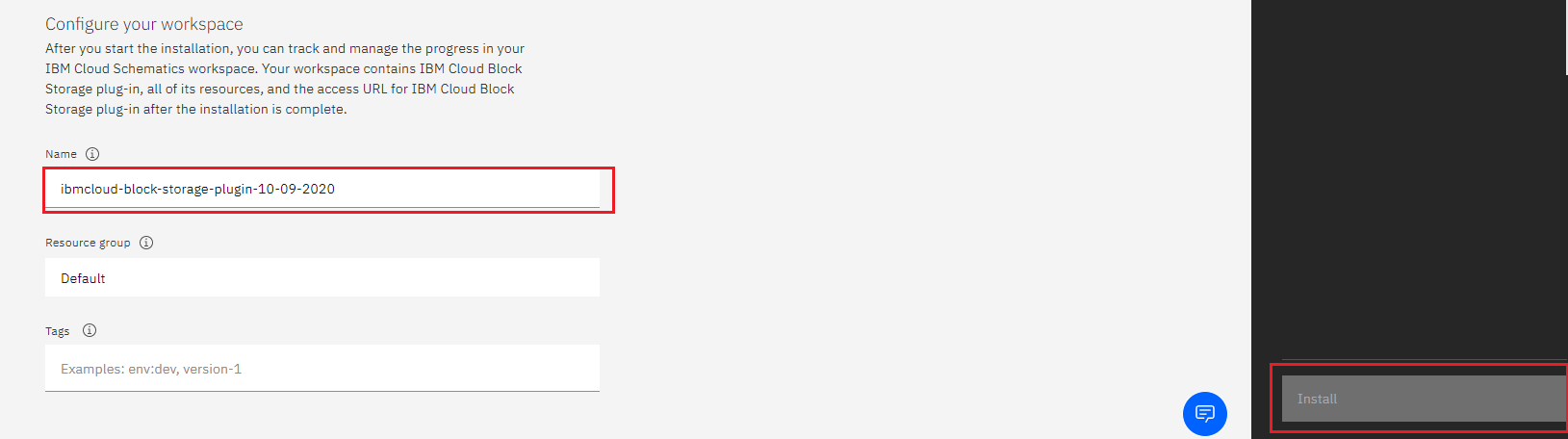
The Block Storage plug-in is a persistent, high-performance iSCSI storage that you can add to your apps by using Kubernetes Persistent Volumes (PVs).

* Click the **Catalog** button on the top
* Select **Software** from the catalog
* Search for **IBM Cloud Block Storage plug-in** and click on it



* On the application page, click on the dot (.) next to the cluster, you wish to use
* Click on Enter or Select Namespace and choose the default Namespace or use a custom  
  one (if you get error please wait for up to 30 minutes for the cluster to finalize)



* Give a **name** to this workspace
* Click **install** and wait for the deployment
* 

**Step 3 For MongoDB**

**Installing using Helm Chart**

$ helm repo add bitnami-ibm https://charts.bitnami.com/ibm

$ helm install my-release bitnami-ibm/mongodb

Specify each parameter using the --set key=value[,key=value] argument to helm install. For example,

$ helm install my-release \

--set auth.rootPassword=secretpassword,auth.username=my-user,auth.password=my-password,auth.database=my-database \

bitnami-ibm/mongodb

The above command sets the MongoDB root account password to secretpassword. Additionally, it creates a standard database user named my-user, with the password my-password, who has access to a database named my-database.

Alternatively, a YAML file that specifies the values for the parameters can be provided while installing the chart. For example,

$ helm install my-release -f values.yaml bitnami-ibm/mongodb

**Tip**: You can use the default [values.yaml](https://cloud.ibm.com/catalog/content/values.yaml" \t "_blank)

Using LoadBalancer services

You have two alternatives to use LoadBalancer services:

Option A) Use random load balancer IPs using an initContainer that waits for the IPs to be ready and discover them automatically.

architecture=replicaset

replicaCount=2

externalAccess.enabled=true

externalAccess.service.type=LoadBalancer

externalAccess.service.port=27017

externalAccess.autoDiscovery.enabled=true

serviceAccount.create=true

rbac.create=true

Note: This option requires creating RBAC rules on clusters where RBAC policies are enabled.

Option B) Manually specify the load balancer IPs:

architecture=replicaset

replicaCount=2

externalAccess.enabled=true

externalAccess.service.type=LoadBalancer

externalAccess.service.port=27017

externalAccess.service.loadBalancerIPs[0]='external-ip-1'

externalAccess.service.loadBalancerIPs[1]='external-ip-2'}

Note: You need to know in advance the load balancer IPs so each MongoDB node advertised hostname is configured with it.

Using NodePort services

Manually specify the node ports to use:

architecture=replicaset

replicaCount=2

externalAccess.enabled=true

externalAccess.service.type=NodePort

externalAccess.service.nodePorts[0]='node-port-1'

externalAccess.service.nodePorts[1]='node-port-2'

Note: You need to know in advance the node ports that will be exposed so each MongoDB node advertised hostname is configured with it.

The pod will try to get the external ip of the node using curl -s https://ipinfo.io/ip unless externalAccess.service.domain is provided.

**Adding extra environment variables**

In case you want to add extra environment variables (useful for advanced operations like custom init scripts), you can use the extraEnvVars property.

extraEnvVars:

- name: LOG\_LEVEL

value: error

Alternatively, you can use a ConfigMap or a Secret with the environment variables. To do so, use the extraEnvVarsCM or the extraEnvVarsSecret properties.

**Sidecars and Init Containers**

If you have a need for additional containers to run within the same pod as MongoDB (e.g. an additional metrics or logging exporter), you can do so via the sidecars config parameter. Simply define your container according to the Kubernetes container spec.

sidecars:

- name: your-image-name

image: your-image

imagePullPolicy: Always

ports:

- name: portname

containerPort: 1234

Similarly, you can add extra init containers using the initContainers parameter.

initContainers:

- name: your-image-name

image: your-image

imagePullPolicy: Always

ports:

- name: portname

containerPort: 1234