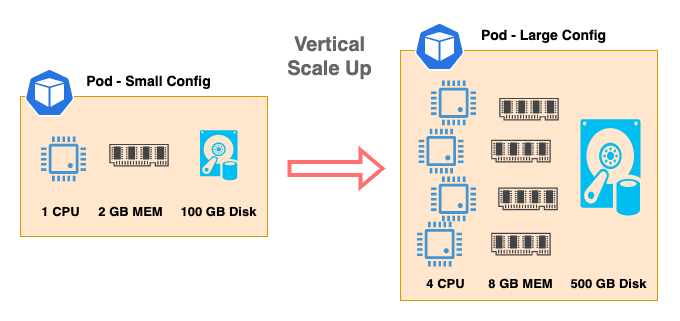


Applications nowadays need to be more agile and flexible to meet increased demands. If you start small with your application deployment and demand increases, your application will ultimately start to max out its resources. As a result, your application response time will increase and cause poor user experience. In order to avoid this situation, you will need to scale your application.

In Kubernetes, a **Pod** is an abstraction that represents a group of one or more application containers. In this article, we will discuss two ways to scale your application pods.

**Scaling Up (Vertical Scaling)**

Scaling up (or vertical scaling) is adding more resources—like CPU, memory, and disk—to increase more compute power and storage capacity. This term applies to traditional applications deployed on physical servers or virtual machines as well.

  
The diagram above shows an application pod that begins with a small configuration with 1 CPU, 2 GB of memory, and 100 GB disk space and scales vertically to large configurations with 4 CPU, 8 GB of memory, and 500 GB disk space. Now with more compute resources and storage space, this application can process and serve more requests from clients.

Scaling up seems to be a good choice if your application only needs to scale to a reasonable size. There are some advantages and disadvantages of scaling up:     
**Advantages**

* **It is simple and straightforward.** For the applications with more traditional and monolithic architecture, it is much simpler to just add more compute resources to scale.
* **You can take advantage of powerful server hardware.** Today’s servers are more powerful than ever, with more efficient CPUs, larger DIMM capacities, faster disks, and high-speed networking. By taking advantage of these ample compute resources, you can scale up to very large application pods.

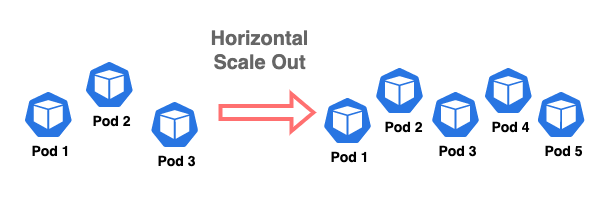
**Disadvantages**

* **Scaling up has limits.** Even with today’s powerful servers, as you continue to add compute resources to your application pod, you will still hit the physical hardware limitations sooner or later.
* **Bottlenecks develop in compute resources.** As you add compute resources to a physical server, it is difficult to increase and balance the performance linearly for all the components, and you will most likely hit a bottleneck somewhere. For example, initially your server has a memory bottleneck with 100% usage of memory and 70% usage of CPU. After doubling the number of DIMMs, now you have 100% of CPU usage vs 80% of memory usage.
* **It may cost more to host applications.**Usually the larger servers with high compute power cost more. If your application requires high compute resources, using these high-cost larger servers may be the only choice.

With physical hardware limitations, scaling up vertically is a rather short term solution if your application needs to continue growing.

**Scaling Out (Horizontal Scaling)**

Scaling out (or horizontal scaling) addresses some of the limitations of the scale up method. With horizontal scaling, the compute resource limitations from physical hardware are no longer the issue. In fact, you can use any reasonable size of server as long as the server has enough resources to run the pods. The diagram below shows an example of an application pod with three replicas scaling out to five replicas, and this is how Kubernetes normally manages application workloads.



Scaling out also has its advantages and disadvantages:

**Advantages**

* **It delivers long-term scalability.** The incremental nature of scaling out allows you to scale your application for expected and long-term growth.
* **Scaling back is easy.** Your application can easily scale back by reducing the number of pods when the load is low. This frees up compute resources for other applications.
* **You can utilize commodity servers.** Normally, you don’t need large servers to run containerized applications. Since application pods scale horizontally, servers can be added as needed.

**Disadvantages**

* **It may require re-architecting.** You will need to re-architect your application if your application is using monolithic architecture(s).

**Scaling Stateless Applications**

Stateless applications do not store data in the application, so they can be used as short-term workers. Kubernetes manages stateless applications very well. In Kubernetes, a HorizontalPodAutoscaler automatically updates a workload resource, such as a Deployment, with the aim of automatically scaling the workload to match demand. This means if the load of application pods increases, the HorizontalPodAutoscaler keeps increasing the number of pods until the load comes back to normal range.

If the load decreases and the number of pods is above the configured minimum, the HorizontalPodAutoscaler instructs the Deployment to scale back down.

The HorizontalPodAutoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The horizontal pod autoscaling controller, running within the Kubernetes control plane, periodically adjusts the desired scale of its target Deployment to match observed metrics such as average CPU utilization, average memory utilization, or any other custom metric you specify.

Kubernetes implements horizontal pod autoscaling as a control loop that runs intermittently. The interval is set by the –horizontal-pod-autoscaler-sync-period parameter to the kube-controller-manager, and the default interval is 15 seconds.

See the link below for more information about Kubernetes HorizontalPodAutoscaler.

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>

**Scaling Stateful Applications**

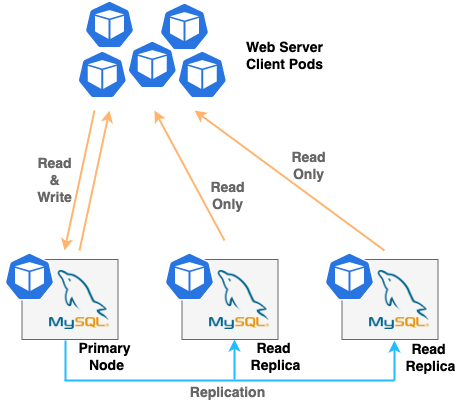
Unlike stateless applications, stateful applications such as databases need to store data in persistent volumes. This makes it more difficult for Kubernetes to manage stateful applications. The storage layer itself is a very complex topic, and we will not discuss it in detail in this post.

For some stateful applications, you may be able to configure Kubernetes’ HorizontalPodAutoscaler to automate scale out operations, but once data is written, it is difficult to scale back down.

For example, if a database gets larger and storage capacity usage gets high, we need to scale up or expand the persistent volume. This operation is usually performed separately from scaling up the compute resource by adding CPU and memory. In this section, we will focus on scale up and scale out of stateful database pods.

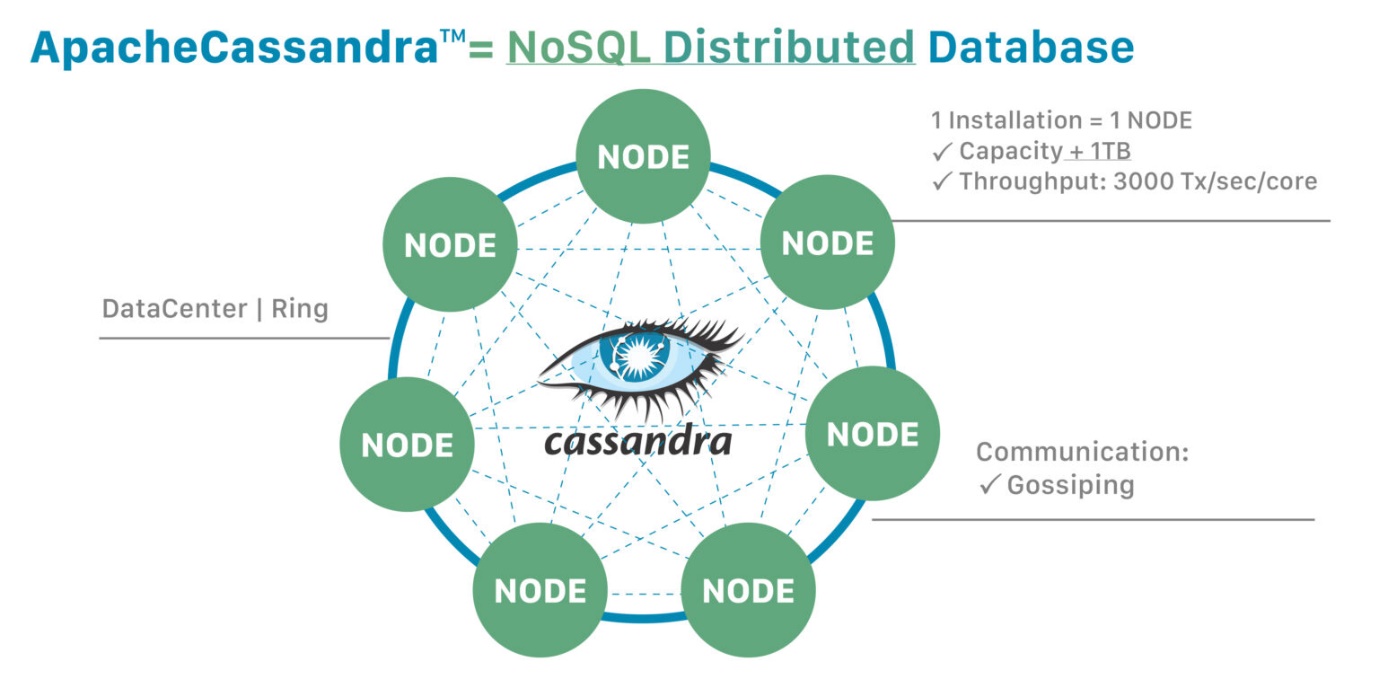
**Scale Up Database Pods Vertically**

When you use relational databases like MySQL or PostgreSQL, you can create multiple nodes of database pods to form a cluster to increase high availability and scale the performance. For example, when you create a 3-node cluster of MySQL, one of the nodes is the primary node that accepts read and write requests. Two other nodes are called read-replicas, and they only serve the read requests. Since you can only write to the primary node, data consistency can be maintained in the cluster.

  
If the demand of read requests increases and write requests stay the same, then you can scale out your database by adding more read replica pods to the cluster. However, if the write requests increase, adding more read replica pods will not help since you can only write to the primary node. In this case, it is much simpler just to scale up your database pod by adding more compute resources. There is a way—called “sharding”—to split a database instance into multiple instances when the database gets too large, but this introduces another level of complexity into the cluster architecture.

**Scale Out Database Pods Horizontally**

If you are using a NoSQL distributed database like Cassandra, you can easily scale out horizontally to meet your demand. Every Cassandra node can perform read and write operations, and this makes a Cassandra cluster a masterless, or peer to peer, architecture. In other words, distributed databases are made to scale very large.



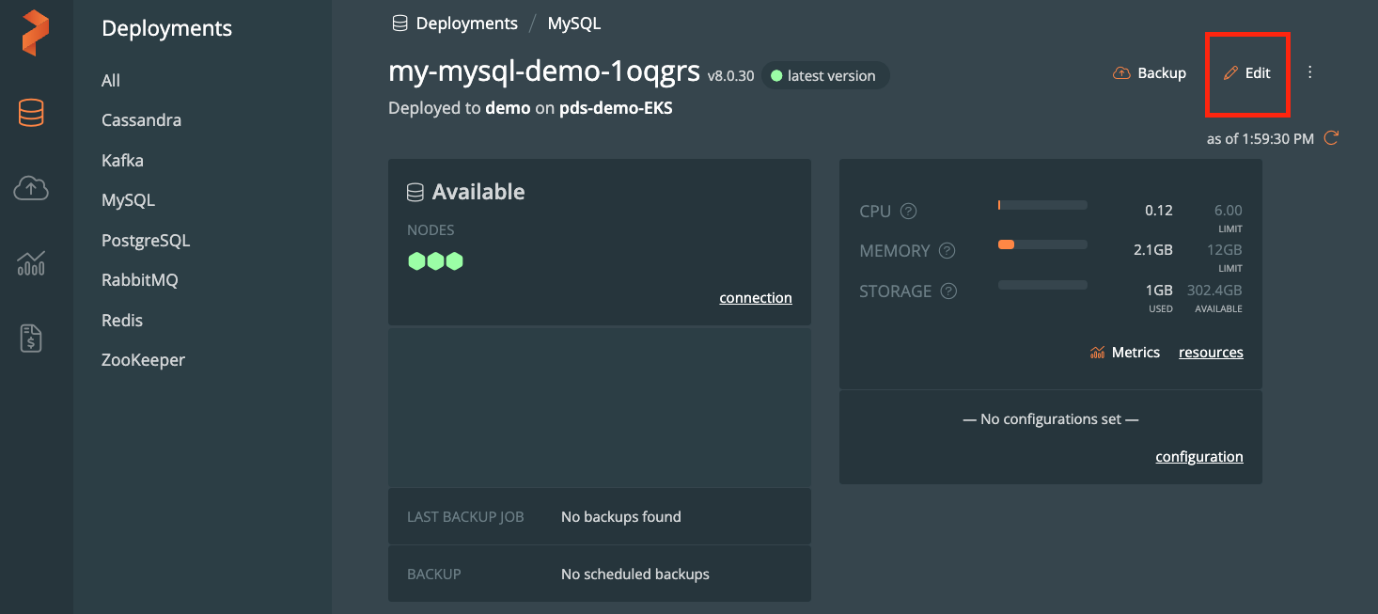
Since Kubernetes usually scales out applications horizontally, Cassandra and other distributed databases are well suited in Kubernetes environments. Adding the pods can increase an application’s TPS (transactions per second) performance and database size linearly. For more information regarding Cassandra, see the link below.

<https://cassandra.apache.org/_/cassandra-basics.html>

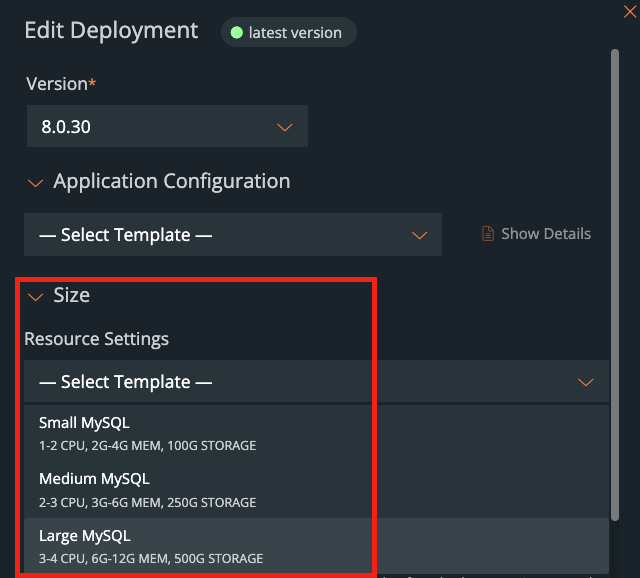
**Scale Up Database Applications with Portworx Data Services**

[Portworx Data Services (PDS)](https://portworx.com/products/portworx-data-services/) allows organizations to deploy modern data services like PostgreSQL, Cassandra, Redis, Kafka, MySQL, Zookeeper, and RabbitMQ on any Kubernetes cluster that is registered to the PDS control plane. Using PDS, developers and DevOps administrators can deploy a highly available database instance on their Kubernetes cluster. PDS also automates Day-2 operations on your data services layer, such as scale up, scale out, backup, and in-place upgrade.

To scale up your PDS database application, log into the PDS console, navigate to your deployment and click **Edit** on the top right corner as shown below.



Within the **Edit Deployment** window, navigate to **Size > Resource Settings** and select the desired compute resource template from the drop-down menu.

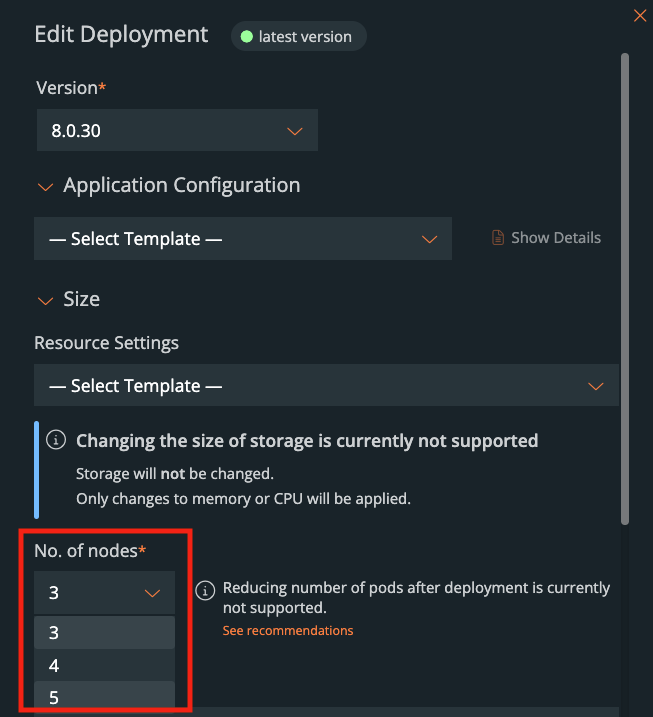


After you click **Apply,** your database pods will be re-created one by one with a new compute resource. Note that storage size won’t be changed with this operation. Only CPU and memory size will change. Separately, you can expand your storage volume manually or use the Portworx Autopilot feature to expand volumes automatically when the capacity utilization reaches a certain threshold. See the link below for more information about Portworx Autopilot.

<https://docs.portworx.com/operations/operate-kubernetes/autopilot/>

**Scale Out Database Applications with Portworx Data Services**

PDS also makes scaling out database applications easy. Within the **Edit Deployment** window, navigate to **Size > No. of nodes** and select the desired total number of pods from the drop-down menu as shown below.



After you click **Apply,** Portworx Data Services will perform the following operations to scale your database pods:

1. Provision a persistent volume with the number of replica volumes specified.
2. Create a new pod and attach the persistent volume.
3. Configure high availability (HA) for newly created database pods. The HA architecture and configurations are specific to each type of application. For example, configuring HA for PostgreSQL is completely different from Cassandra HA configuration. PDS performs this complex HA setup for you automatically.