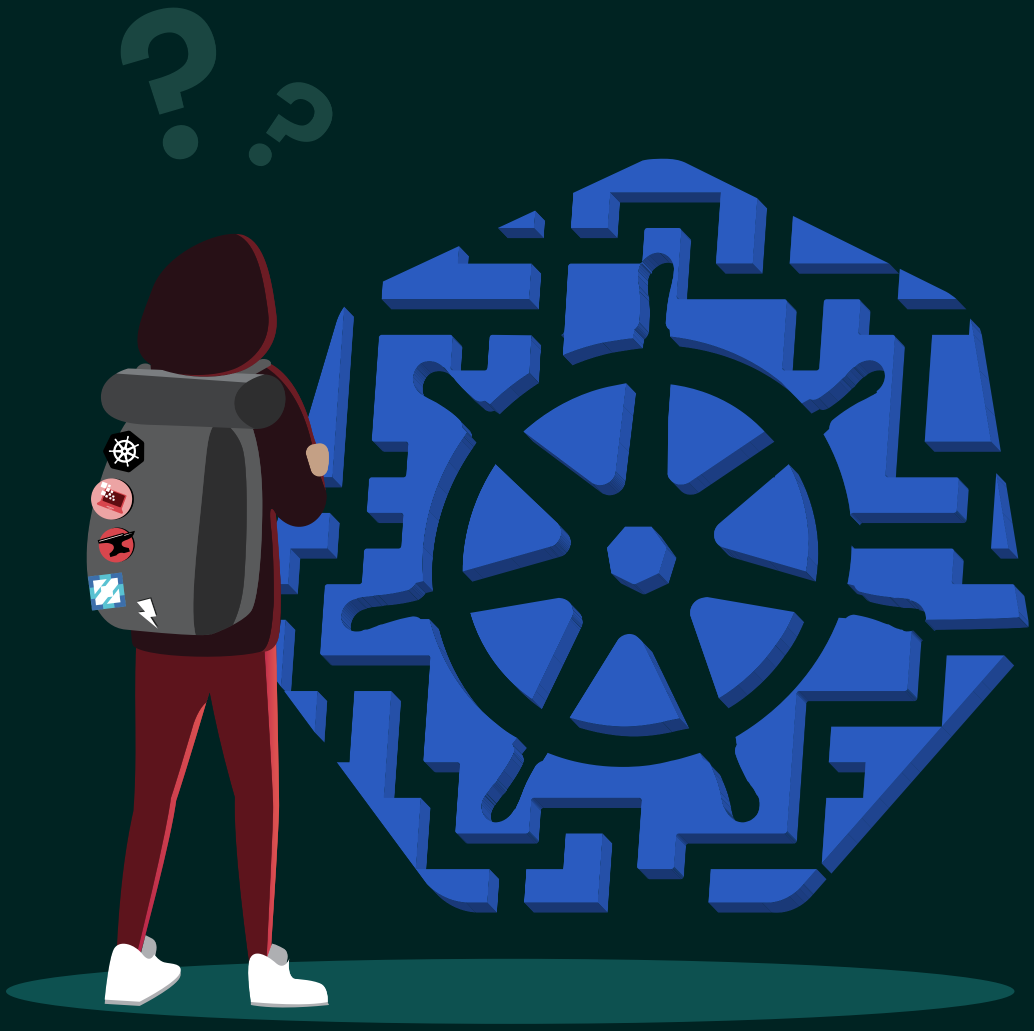
StormForge

Harnessing the Complexity of Kubernetes to Benefit Your Business



What's Inside

Your Go-To for Agility

Kubernetes (aka k8s by many). There has been a great deal of hype around this technology. In fact, **Gartner's latest hype cycle for I&O automation** (July, 2021) shows container management, which is the primary function of Kubernetes, is now past the "peak of inflated expectations" and starting the slide into the "trough of disillusionment."

While the realities of Kubernetes complexity are setting in, there is good reason for the hype. Cloud native architectures running on Kubernetes holds great promise for:

- Faster time-to-market
- Superior user experience
- Easier scalability
- Improved cost efficiency
- Always-on availability

So, why the slide into the trough of disillusionment? As IT organizations dive deeper into Kubernetes, we hear a number of frustrations, including:

- It's too time consuming.
- It's too complicated.
- It's too expensive to implement.
- We don't have the resources.
- Our staff doesn't understand the technology.

Before your company abandons Kubernetes, let's take a deeper look at what this technology has to offer.

The Road to Kubernetes Success is Fraught with Peril

While Kubernetes and containerization hold great promise for increasing business agility and improving resource efficiency, complexity makes it hard to realize these promises. Here are just a few examples of the real-world challenges that enterprises face. You can find many more at **k8s.af**.

airbnb

"During our effort to eliminate waste, we found a number of large services not using horizontal-pod-autoscaler (HPA), and services that were using HPA, but in a largely sub-optimal way..."

Your Business, Your Benefits

Let’s start with some of the basics of Kubernetes.

First, you must know what a container is. In terms of technology, a container is an application that has been packaged together with all of the necessary code and dependencies such that it can be deployed on almost any platform (so long as said platform includes a container runtime engine). Containers make deploying apps and services incredibly easy. And because they are portable, a developer can create a containerized application with a development environment (such as their desktop computer) and then move it to a production environment, where it should run as expected.

Kubernetes is much more than that. What Kubernetes does is make it possible for developers and admins to manage numerous aspects of deploying and running containers. With Kubernetes it is possible to orchestrate application deployment, scaling, and management—all from a single point of entry.

Think of it this way. Imagine you have a massive collection of Legos — different colors, shapes, and sizes. Alone you’ll eventually be able to take those individual bricks and put them together such that they create a very specific sculpture. However, without guidance that could take some time and you might not get the exact end results you were hoping for. Kubernetes is like the photo and the instructions you get with those Legos—it makes it possible for you to piece those bricks together, in exactly the right configuration, and do it in a much more efficient and reliable way.

However, the problem with that analogy is that it makes Kubernetes seem really easy. Up to a point it is. That Lego photo might give you the basic idea of what you’re building, but there are intricacies and inner workings it might not show.

Plus, Kubernetes isn’t about building Death Stars with Legos. It’s much more.

New Relic.

“After we had the application deployed in Kubernetes and routed some production traffic to it, things started looking worrisome. Request latencies in the Kubernetes deployment were up to x10 higher…”

Grafana Cloud

“Grafana Cloud experienced a ~30min outage in our Hosted Prometheus service. To our customers who were affected by the incident, I apologize… There were not enough resources on the Kubernetes cluster to fit the new Cortex cluster…”

What Makes It Special?

The biggest promise that Kubernetes makes is that it makes it possible for your company to get the absolute most power, flexibility, and reliability from your container deployments. With this particular management platform, you can develop a very specific application (or set of applications), deploy them to a cluster, and then set them to scale as needed so your business benefits from a level of failover and reliability you've probably never experienced. And, with the right tools, much of this can be done automatically.

That's a big deal, especially in today's ultra-competitive world of business, where the slightest hiccup in your system can cause your company to fall behind. And with Kubernetes, there are a lot of "moving pieces" that can go wrong.

Deploying Apps the Containerized Way

The building blocks of Kubernetes are numerous and can be complicated. Here's a short list of some of the components that come together to help form a Kubernetes deployment:

- **Cluster** - The cluster is a collection of Kubernetes servers (be they bare metal or virtual) that work together such that if one node fails, the deployed application will not go down.

- **Kubernetes Controller** (once called the "Kubernetes Master" but that nomenclature is thankfully changing) - The Kubernetes Controller is the one server in the cluster that is tasked with managing all nodes and various core controllers.

- **Worker Node** - This is a single server (either bare metal or virtual) that serves the cluster. All containers are deployed to the nodes (with the controller handling the orchestration).

- **Pod** - Pods are the smallest deployable units that can be managed by the controller. A pod consists of one or more containers that share the same IP address and port.

- **Namespaces** - Namespaces make it possible to divide resources between users without running into name collisions. However, while namespaces do provide some segregation, they are not a complete tenancy solution.

- **API Server** - This exposes APIs to the cluster.
- **Scheduler** - The Scheduler watches for newly created pods and selects a node for them to run on.
- **etcd** - This component stores the configuration information used by Kubernetes to track information about all of the objects in play in a cluster.
- **Docker/CRI-O** - This is the container engine that makes it possible to deploy containers.
- **Kubelet** - Kubelet is the agent that runs on each node in the cluster.
- **Kubernetes Proxy Service** - This is a service that runs on each node and makes it possible for containers within the cluster to be reached from outside (from LAN or WAN).

And that's just the short list. There are quite a few more pieces that make up this rather complicated puzzle. And, to make matters even more complicated, many of these pieces need to be configured manually. And within each piece, there are sub-pieces that must also be configured.

So within a Kubernetes cluster, there are seemingly countless opportunities for optimization, each of which will affect your cluster in different ways.

The Complexity of It All

Now that you understand the complexity of Kubernetes, let's make it even more so. Assume you already have your Kubernetes cluster up and running (that is, after all, the easy part). Now you must develop, build, and deploy an application and/or a service to the cluster. How do you do that?

You craft a YAML file that declares every aspect of your app or service and then, using a command like helm (package manager for Kubernetes), you deploy the service to the cluster. What is a YAML file? A YAML file is a configuration file, written in a human-readable, data-serialization language.

Once you've pieced together your YAML file, what do you do? Within the Kubernetes controller, you'll find a very powerful command, called kubectl. That command enables you to deploy to your cluster. Say, for instance, you have a YAML file named deploy.yaml. In that file you have everything put together to deploy your application. How do you deploy it? Simple. From the command line you'd issue something like:

```
kubectl create -f deploy.yaml
```

It really is that simple. But what if you find you need to change a configuration or two within your YAML file? The kubectl command has you covered. After you've made your edits, apply them with the command:

```
kubectl apply -f deploy.yaml
```

Kubernetes also has another very useful tool that can help make your life a bit easier. This tool is called Helm, which is a package manager for Kubernetes. Helm streamlines the installation and management of Kubernetes applications.

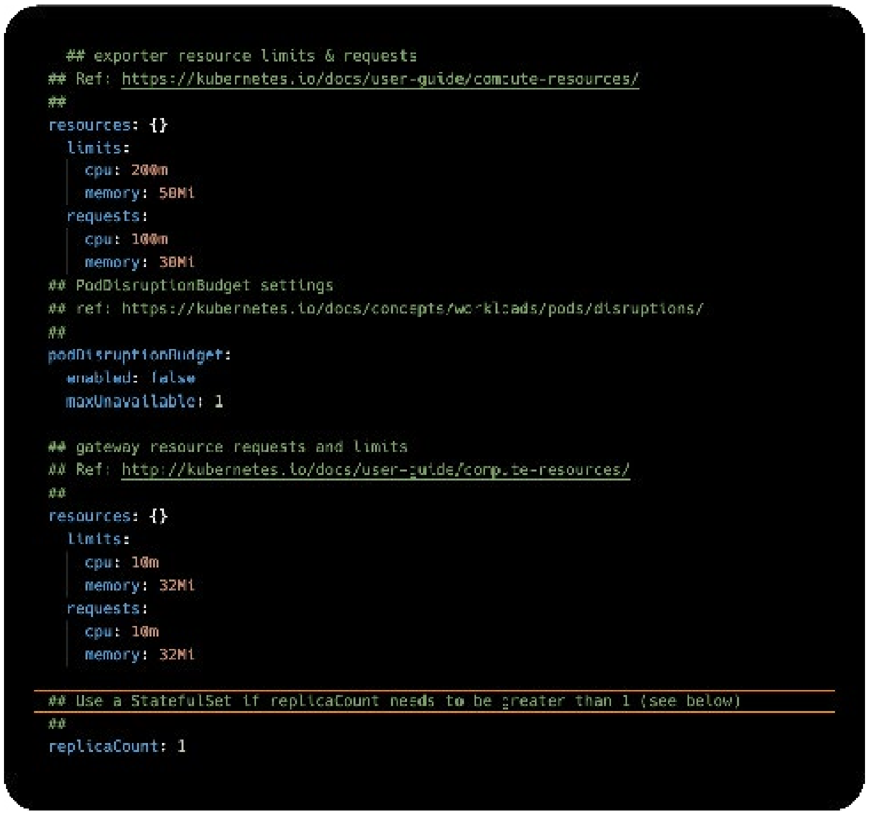
Once you have Helm installed, you can then work with charts, which are a collection of files that describe a related set of Kubernetes resources. With these charts you can deploy simple or very complex apps and services to your Kubernetes clusters. For example, with Helm you could quickly download the Prometheus chart with the command:

```
helm install stable/prometheus --generate-name
```

That command would download the chart for Prometheus and save it to your local storage. You could then modify the chart to perfectly fit your needs.

Why reinvent the wheel? With the help of Helm charts you can deploy incredibly complicated services and apps to your Kubernetes cluster with minimal effort.

These YAML files become even more complicated, because there are so many options that can be defined. Such a configuration could look like this:



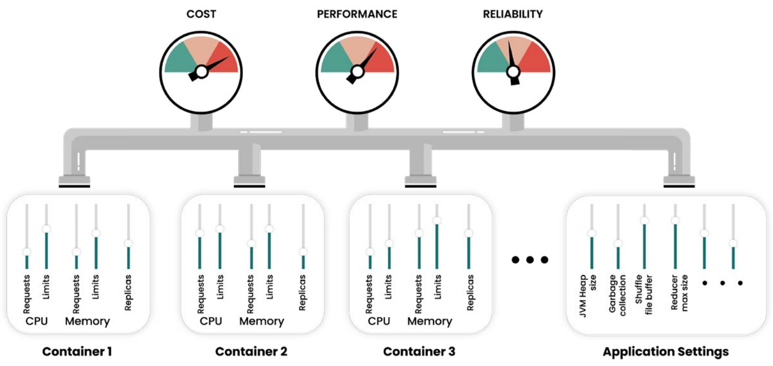
```
  ## exporter resource limits & requests
## Ref: https://kubernetes.io/docs/user-guide/compute-resources/
##
resources: {}
  limits:
    cpu: 200m
    memory: 50Mi
  requests:
    cpu: 100m
    memory: 30Mi
## PodDisruptionBudget settings
## ref: https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
##
podDisruptionBudget:
  enabled: false
  maxUnavailable: 1

## gateway resource requests and limits
## Ref: http://kubernetes.io/docs/user-guide/compute-resources/
##
resources: {}
  limits:
    cpu: 10m
    memory: 32Mi
  requests:
    cpu: 10m
    memory: 32Mi

## Use a StatefulSet if replicaCount needs to be greater than 1 (see below)
##
replicaCount: 1
```

The above example is very basic, just a few select lines from a 1600 line example of a popular solution. Now imagine you have a very complex Kubernetes configuration file that includes a number of application definitions, each of which defines resource usage specifications. Such a YAML could wind up with hundreds of definitions. Crafting such a file is challenging, **but not nearly as challenging as optimizing that configuration** to not only ensure your application or service performs well enough to meet your SLOs and SLAs, but also to make sure that the application runs cost-effectively. This is especially important when your Kubernetes deployment is on a public cloud, where you are charged based on the resources you use.

Benefiting from AI and ML

Consider this: You have crafted an incredibly complex YAML file for the deployment of a crucial application to a Kubernetes cluster running on AWS. However, the application isn't running efficiently or effectively. You open the YAML file and your first inclination might be to increase the RAM and CPU resources that are available to the application. You re-deploy, only to find the application still isn't running properly. What do you do? Wash, rinse, repeat. In the end, you could wind up with a YAML file that hands over a large amount of resources to your applications. That equates to an exponentially more costly deployment.

When your Kubernetes application configurations (such as startup parameters, environmental variables, Java JVM tuning) grow exponentially in size, the idea of optimizing them becomes challenging. With dozens, or even hundreds, of configuration options, each of which you can "dial in" to help improve your deployment, it can be almost impossible to know which option(s) to change.



With dozens, or even hundreds, of configuration options, each of which you can "dial in" to help improve your deployment, it can be almost impossible to know which option(s) to change.

You can increase the CPU for the database component, only to find it makes little to no difference. Go back and up the memory for the web server. Nothing. Change the CPU for the database back to its original value and up the memory option. Still no change. Increase the CPU for the web server and you might see a slight uptick in performance. However, what you might not see is that all of a sudden you've exceeded your resource limits your provider offers and will be billed for the overages.

Back to the drawing board.

Had this been a simple Kubernetes YAML file (say with only a limited number of components and few options), this back-and-forth optimization might not take so long. But when those YAML entries number in the hundreds, you're looking at an incredibly time consuming (and frustrating) endeavor. That's how complicated a Kubernetes deployment can become.

But what if you could use Artificial Intelligence and Machine Learning to test your configurations and perfectly tune your application such that it will not only run with a high level of efficiency but also with a low cost of delivery? If that were possible, why would you trust that optimization to first-generation management tools that are both imprecise and manual? Such tools often miss opportunities for huge performance gains and cost reductions. Think of it this way: Your deployed application is like a living, breathing animal and each component of that application is critical to the life of that animal. What is the long-term impact of that animal not getting enough water, food, or sleep, or carrying around too much weight or having a buildup of some harmful chemical in its system? Diagnosing each one of those components individually can be pretty simple. But because they are interrelated and work in harmony, the complexity increases exponentially.

Believe it or not, such a service is available. **StormForge** uses machine learning to optimize your Kubernetes configurations in ways the human mind is simply not capable of doing. In a matter of hours, StormForge can help you find the perfect configuration that will have your deployment humming and your bottom line smiling.

K8s is Great

There are a number of ways to make Kubernetes a part of your business. How you do so will depend on the scale at which you plan to deploy. If you're a small company (or a developer) looking to kick the tires of this technology, you can always deploy a Kubernetes cluster to your data center. This would require a minimum of 3 servers (be they bare metal or virtual) and could serve as a solid development environment to create the apps and services you are looking to deploy.

If, however, your company exists within the realm of enterprise, chances are pretty good you'll need the ability to seriously scale to meet the ever-changing demand your business will surely experience. If that sounds like you, your best bet would be to sign up for the likes of Amazon Web Services, Azure, or Google Cloud. These services make it easy to deploy a Kubernetes Cluster and scale it out to meet (or exceed) your needs.

Just remember, when you're expanding your container deployments to scale, you're going to want to ensure those configurations are optimized for performance and cost effectiveness. Don't just look at it from the myopic view of performance, as there are other considerations that can help make that containerized application serve you better.

Request a demo of StormForge to see how it can help in your environment.

Discover the Advantages of StormForge

Let our experts assess how StormForge can deliver the foundation for your Kubernetes success.

Schedule your demo today.

REQUEST A DEMO →

StormForge