nine
cloud navigators

Getting started with Kubernetes

10-point checklist

Getting started with Kubernetes 10-point checklist

1. Have you automated your container builds and pushes?

Kubernetes requires that your image is fetched from either a private or public container registry, so if you haven‘t already automated your testing and deployment pipeline, pushing your images automatically after committing to your git repo might be a good place to start! Chances are you’ve already built and published an image to a registry, so the commands you need to run will be familiar and you can focus on finding the right tools for automating your deployment workflow, starting from the image push you can work backwards, considering the build, application testing, code-review and any other workflow steps that you may need to implement.

2. How lightweight is your container? Have you cut out everything you don’t need?

Your applications container image is going to get pulled by various systems throughout your pipeline, it may be needed for testing, for integration deployments, by a CI/CD system and by other systems. To optimise both the time it takes to start a new version of your application and the cost of bandwidth you should aim to remove everything from your container that you don’t need. This may even mean using specifically container optimised operating systems in your container to further reduce the container size. In addition, reducing the number of layers of your images will decrease the sum of attack vectors, increasing the security. Thus, we highly recommend making your containers as light as possible, e.g by using multi-stage builds.

3. Have you written your Kubernetes configuration?

If you haven’t yet written your Kubernetes manifests then you should consider the various options for templating languages. If you have experience using docker-compose you may consider using Kompose to translate this file into the Kubernetes configuration that you need, and this can be a good starting point if you are just beginning to work with Kubernetes. However, it’s important to understand that your Kubernetes configuration is a vital part of your application, and should be treated as such, make sure you understand every line.

Kubernetes uses a declarative configuration and there are a number of templating languages to help you generate the sets of files that you need, such as Jsonnet.

You may even wish to investigate more complex languages such as Cue, Pulumi or cdk8s, which provide the opportunity to define configurations in code and output in formats suitable for working across different clouds, languages and tooling.

4. Have you packaged your configurations in a Helm chart?

Once you begin to look at how to deploy dependent services that your application needs, even if just for testing, you are almost bound to come across Helm charts. A Helm chart is a collection of files that describe a related set of Kubernetes resources, which could be anything from a simple service to a full-blown web application stack. As such you’ll need to at least familiarise yourself with the concepts behind them, but it’s probably a good idea to make a helm chart for your application since it simplifies the deployment of complex applications, and Helm is ubiquitous in the container space.

5. Do you know your application’s performance profile?

To get the most out of Kubernetes you need to understand how your application performs under load, and what are usual and unusual performance indicators. By knowing at least the CPU and RAM profile of your application you will be able to properly configure your resource requests and limits. Configuring these properly helps to optimise your cost but also helps for the self-healing of your application. Without proper configuration, your application might not restart if there is a problem or alternatively enter CrashLoopBackOff if Kubernetes sees normal application behaviour as breaching your resource requirements. However, it is highly unlikely that you will get the performance profile right the first time. Therefore, it is suggested to monitor the actual usage of your application and constantly adjust the requested resources.

6. *Have you considered how your application scales?*

One of the most important and powerful aspects of Kubernetes is to easily scale your containers horizontally. Once you know the performance profile of your containers, you should consider using a HorizontalPodAutoscaler to scale your application depending on the current traffic. Gone are the days of fearing high traffic peaks. Instead, let the Kubernetes cluster work for you.

7. *Have you considered how your application responds to restarts?*

Kubernetes is going to start, stop and restart your containers all the time and that's a feature, not a bug! It will try to restart your application if it thinks it is failing, or it will move your application to a different node if some maintenance work is being carried out, if you change your node pools, or if you need to scale up/down, or many other things. So you need to think about what happens when your application restarts, and make sure that it's not doing things like running database schema upgrades every time or performing any other blocking actions which might impact the ability of your customers to reach your application. You'll need to think about this for scaling too since you'll want instances to start and stop as quickly as possible to get the most from the dynamic capabilities of the Kubernetes scheduler.

8. *Have you considered the availability of your application?*

Physical or virtual machines fail and they will keep failing. There is no way around it. However, you can set up your containers to eliminate any single point of failure, also known as high availability. There is little point in running 10 replicas of your containers on a single node if you have a multi-node cluster. Therefore, you should spread your replicas across as many nodes as possible and increase the robustness and availability of your application.

9. *Have you considered how your application responds to unavailable services?*

You might want to think about your service stacks‘ general approach to service unavailability as well. Unavailability is always going to be a possibility regardless of the infrastructure, service provider or architecture you choose, but it's especially important in cloud environments. So ensuring your application is resilient in such scenarios, that you remain accessible and don't lose data, should be of primary concern in your application development phase.

10. *Have you considered how to run the services your application depends on?*

There is a lot to think about when running your application in Kubernetes, and though it can bring massive benefits when used properly it can be somewhat overwhelming to deal with everything. Regardless of whether you are new to Kubernetes or not, it may be a good idea to partner with an experienced third-party provider who can offer auxiliary services of a guaranteed level of quality to use with your application. If you are experienced you might want to go directly to a cloud service provider and simply consume their services, but if you need more than this, such as direct contact with engineers, additional services, solution architecture support, or something else, engaging with a service provider such as Nine, with years of experience running both traditional and container infrastructure could provide significant benefits.