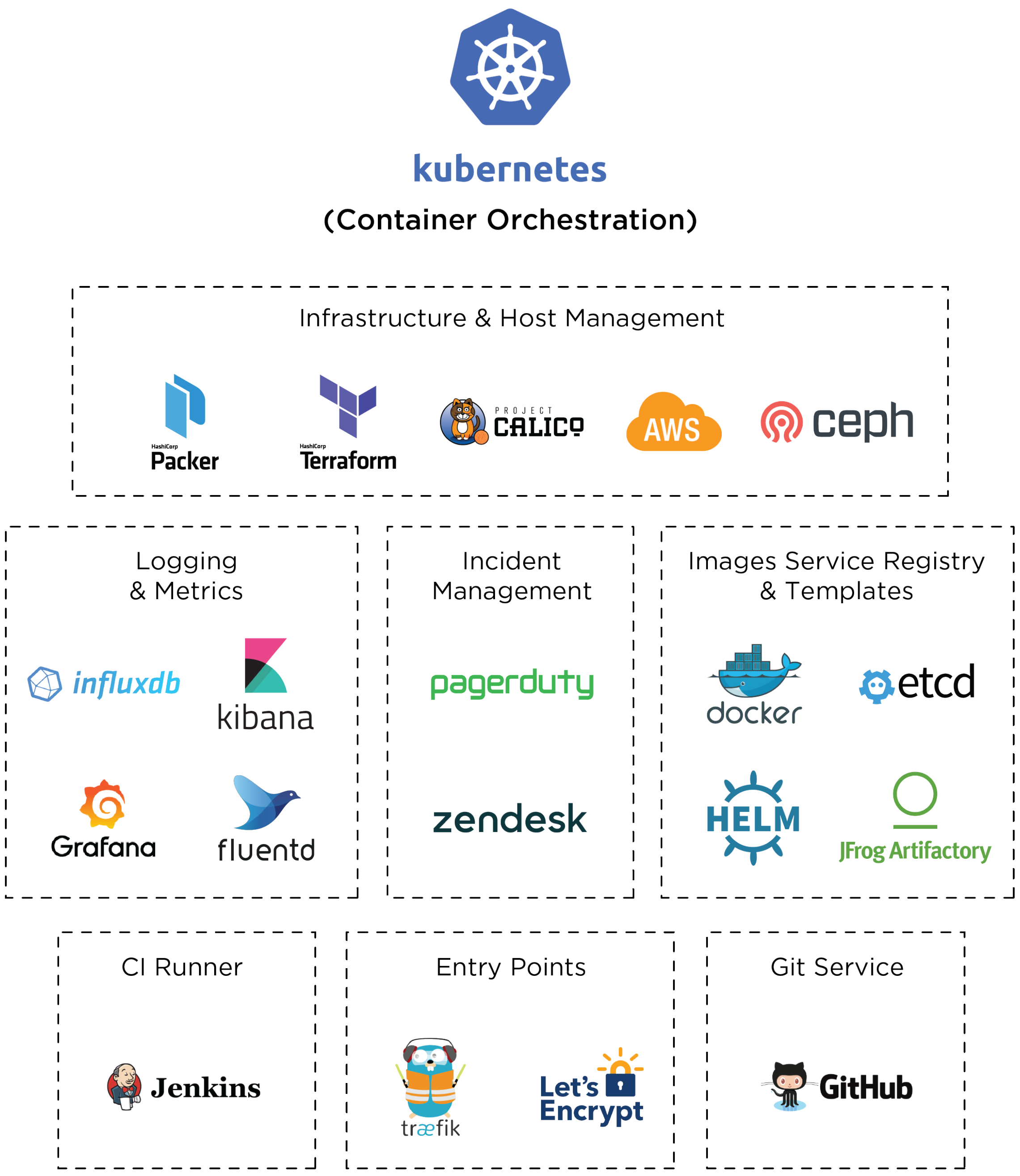
An Alternative to Kubernetes
The Platform.sh PaaS



And you are still not getting

- High-Availability
- Security Updates
- Backups
- Automated generation of staging clusters
- Environment Cloning
- A web Application Firewall
- CDN

Introduction

This document has been written for the Enterprise management reader who is experiencing anxiety about starting, or continuing a Kubernetes project, and wants to understand the likely costs, risks, and alternatives available to take advantage of containerisation and its benefits to their organisation.

Essentially, we are talking about buy versus build. You might use a framework such as Kubernetes to build your container based architecture, which is complex and expensive, and at the other end of the difficulty spectrum, you might just buy a PaaS.

For your information, a PaaS will support many different technologies out of the box, and will allow developers to concentrate on coding whilst the PaaS handles test environments, deployments to production, resilience in the live service, scaling etc…well, some of the better ones anyway.

When considering Kubernetes in the first place, a major distraction for many managers can be Docker itself, so it's important to understand that it's not the only container technology available, even though it's currently the most popular. However, the domain is fast moving, and much of what was **the-next-big-thing** 6 months ago, has already become *'less interesting'*.

Of even greater importance, is that it's not really about whatever container technology your technical team is excited about, it's actually about a better way to manage what you put inside the container, namely the applications you want deployed and the dependencies they rely on. And whatever big effort you decide to spend time and effort on needs to have big benefits for your business, which in this case means a better experience for all the stakeholders involved, those being developers, operations, service management and the end customer.

This document is based largely on the experiences of Platform.sh PaaS customers who are migrating away from Kubernetes.

Management Summary

Although Kubernetes comes with ultimate flexibility, you need to build it yourself, whereas a PaaS introduces many efficiency gains that reduce the cost and complexity of container based hosting in the cloud, accelerates application delivery, and enables new service propositions quicker. Specifically, significant PaaS savings relative to Kubernetes would be:

1. Avoiding the cost and management overhead associated with building and running Kubernetes, including the cost and complexity of multiple vendor management tool sets, most of which only allow your engineering teams to execute manual tasks quicker, as opposed to effectively automating critical blocks of business process.

2. Taking advantage of the cost economies of scale that a PaaS vendor can offer around compute and storage resources, which can amount to half of what you are paying your hosting vendor direct.

Building your own Kubernetes infrastructure is 5-8 man years of effort and will cost you upwards of $1.5m, to achieve just a thin layer of container management for a limited set of technologies, benefiting some aspects of development only. Operating and maintaining it will then cost you a minimum $500k per annum.

Compared to a market leading PaaS, you will have about 30% of the value for developer workflow, about 40% of the value for infrastructure management, and no SLA for your production services.

Based on our own investment and running costs, allowing your Kubernetes build team to bridge the gap to a PaaS such as Platform.sh, would take you 40-50 man years of effort, cost you 10 times as much to build, and several times as much to operate.

What does Kubernetes really give you ?

Are you sure there's not a better way?

Kubernetes is an incredible piece of software, but also an incredibly complex system. It gives you the flexibility to run any sort of container you want, but your infrastructure team will still need to solve many problems around the configuration and ongoing management of those containers and their associated Continuous Integration (CI) development and deployment process workflows. This all requires a lot of effort to build in the first place, and is then very hard to maintain. If you thought you were going "all standards", the reality is that you are actually embracing a never-ending regime of DIY.

And that just covers the application build/test/deploy process. You may well need a costly Managed Kubernetes Service to then manage those services in the production environment to ensure they all stay up and running.

Kubernetes flexibility is actually Kubernetes complexity

Running vulnerable code in a container is no safer than running it in a Virtual Machine (VM), and running vulnerable services (eg. database) is no easier when they're inside a Kubernetes cluster.

Docker and Kubernetes expose interfaces that look like abstractions and simplicity at first glance, but what you're actually buying is a bulk load of nuts and bolts, which means complexity and therefore risk. What looks like flexibility and control, soon becomes very expensive to maintain and your costs may well skyrocket if you have to achieve your original objectives.

You can put anything in your (Docker) container

Docker is based on images, and there are tens of thousands of container images available. Trouble is, an image is the result of a script, so what's actually in the image is often questionable. In fact the contents of an image are so opaque that they are commonly known as blobs.

Docker makes it easy to package up something yourself to run, but unless you're building all your containers from source, they then become difficult to maintain. And if you are using blobs from elsewhere, how can you trust they don't contain vulnerable libraries and executables, or whether or not they are being maintained properly?

No reproducible build chain or read-only infrastructure - which will take you many man years to build

Unless you invest in building a "*reproducible build chain*", you are running services you may not always know how to update, because once you deploy your container, there is nothing to prevent it from changing - it's just a normal program, and can write to disk and even update its own code.

So, you need to ensure you have a repeatable build process with deterministic results, but also an *"immutable read-only infrastructure"* to make sure once a container gets into production, nothing can ever change it whilst it's there.

These two items alone can take several man months each to plan, and many man years to build.

Not enough intelligence in the cluster orchestrator - 2-4 man years to build

Because you wanted something flexible enough to run any container, you now have all the uncertainty highlighted above, and this makes it very difficult for the cluster orchestrator to know what each specific process is doing at any one time. Unless everything you are running is stateless, allowing the orchestrator to abruptly shut down a container whilst writing to disk means certain data corruption, and a catastrophic impact to the application. This is a key component of container failover and therefore application resilience, and will take 20-50 man months to build out properly.

So, do you really need all this flexibility?

There are a lot of advantages to standardising on fewer technologies, and making your process management less project specific. You just don't need to approach every new project with a blank sheet of paper, and design everything around it in deeply specific detail, which will inevitably be a bigger overhead to maintain. The majority of your projects should be as standardised as possible, with only a handful of exceptions.

And are your requirements really outside the 99% of what most other companies are doing ?

What you are standardising on is also critical here. Kubernetes allows you to build and support any possible back-end service, especially important when your requirement falls outside the 99% of most commonly required services. If you think you could make do with the 99% of what's out there and already available, you probably don't need these extreme levels of flexibility.

Considerations when you build out Kubernetes

What are pitfalls of Kubernetes?
What are the right questions to be asking your technical management teams when deciding on the best strategic course to set

Kubernetes may seem like an off-the-shelf product, because there are many artificial use cases which are extremely easy to show off, and some of the higher-level distributions really demonstrate its power as a tool. But it's still a very raw tool.

Again, much of what we cover in this document, we have learned from our customers migrating from Kubernetes to the Platform.sh PaaS.

What an implementation will and won't give you after several man years of build effort

Building a basic Kubernetes/Docker implementation yourself could take 5 - 8 man years of experienced engineering developer effort, to achieve just a '*thin layer of containerisation process improvement'*, for developer workflows only, and that support a limited number of technologies.

You still need a lot of DevOps and scarce knowledge to run it

Ongoing day-to-day operations will require a team of DevOps and Support staff and will still include various manual activities such as configuration management, transferring data between environments for testing and so on. Investigating and repairing runtime issues with a complex Kubernetes cluster requires everything from specific Kubernetes semantics knowledge, Docker knowledge, networking and storage knowledge all the way down to Kernel specific behavior; all of which are changing very rapidly! As an example of the level of complexity your internal users are going to be exposed to, please take a glance at some of the actual administrator documentation:
https://kubernetes.io/docs/concepts/cluster-administration/sysctl-cluster/

A great developer experience will still be a long way off

Multiple development environments for parallel feature development workflows (ie. replicating a PaaS like experience) will be very expensive due to needing multiple dedicated test servers, unless delivered with cloning technology in the Cloud. This is where the vast majority of developer gains will be achieved, in terms of accelerating the velocity of new features into production. Going back to the Kubernetes cluster orchestrator that we talked about earlier, without a fine-grained vision of the status of *"stateful services"*, performing any operation that requires consistency will be very hard without stopping the service. To do this properly, you need native copy-on-write support and immutable containers, without which this sort of operation is going to be hard-to-impossible to achieve with any degree of safety plus performance.

The patchwork of underlying services to manage spells danger

However you build out your Kubernetes implementation though, you will still have a patchwork of underlying 3rd party services to manage, which means many single points of failure (SPoF), frequent release management, incoherent roadmaps, additional costs (vendor cost for your choice of toolsets), no scale economies against actual compute/storage resources consumed, additional pressure on - and expertise required in - the internal support function, DevOps expertise and retained knowledge practices including ongoing documentation and staff retention.

No SLA in production, and no high availability (HA)

And to put the cherry on the cake, a robust production service SLA is probably not achievable either – and certainly not high availability (HA) - without many more man years of effort. The same is true for resilience in the persistence layer, and shared cluster management. This is because Kubernetes was not originally built for persistent services, in fact it has a number of built-in assumptions about all external data services coming with HA included.

Furthermore, any Kubernetes feature not related to running stateless application services has only been recently added in, and as an afterthought it seems. Basically, if your whole infrastructure wasn't designed as *"cloud native"*, you are out-of-luck.

High initial build investment plus high running costs

For the basic Kubernetes reference case described above, your 5 man-year build cost will approximate $1,350,000. And you will have something to show for this; your development teams will have a better development experience, and your DevOps team will have an easier time managing the infrastructure, but you are unlikely to have an end-to-end SLA across your applications, and definitely no HA.

total cost for basic kubernetes implementation

					140%	
	Type	**FTE***	**Years**	**Salary**	**Fully Burdened****	**TOTAL**
BUILD	**Developer engineer**	1	3	$70,000	$98,000	$294,000
	Developer enginner	4,5	1	$90,000	$126,000	$567,000
						$861,000
OPERATE	**Developer DevOps**	2,5	1	$90,000	$126,000	$315,000
	Support	2,5	1	$50,000	$70,000	$175,000
						$490,000
TOTAL						$1,351,000

Diagram above showing high level Kubernetes build and run costs.

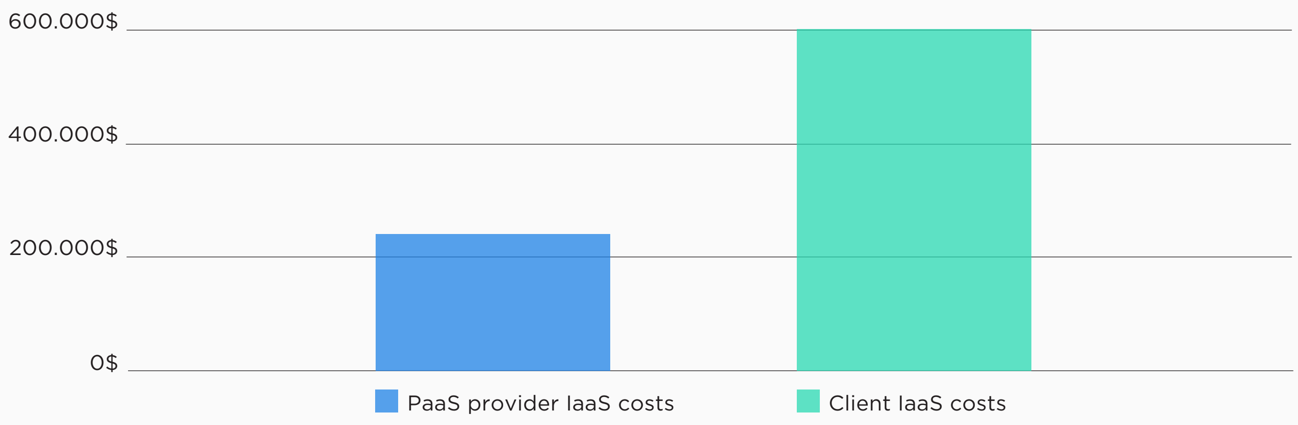
*Notes: *FTE (Full Time Equivalent), **Fully Burdened includes salary plus 40% for office/employment overhead*

Annual support cost plus ongoing development will likely exceed $500,000.

Costs of compute and storage resources way higher than a PaaS alternative

In addition to the initial build and run, you will likely be buying IaaS/hosting resources at 2-3 times the cost of a PaaS provider, due to **a)** the volume discounts they will be receiving, and **b)** the higher densities they will be achieving through balancing larger container workloads. Based on annual usage of 500 cores, 1TB of SSD storage and 50 Tb of bandwidth, we estimate your costs with one of the major global hyperscaling IaaS providers to be in the region of $600,000 p.a., compared to roughly 35-45% of that through a PaaS provider with a few thousand customers.

IaaS costs for Compute Storage Bandwidth



Considerations when you build out Kubernetes

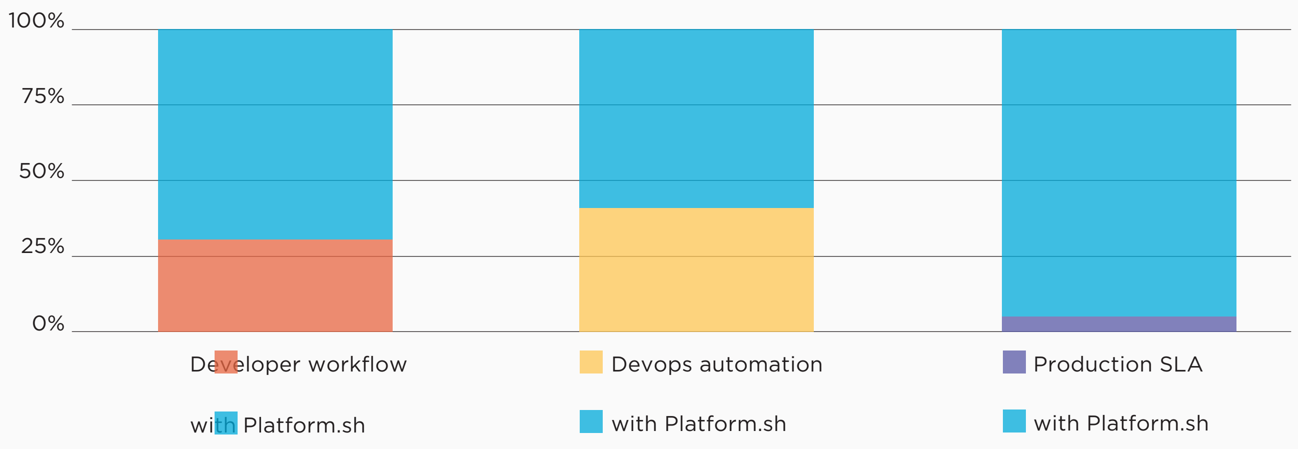
Cost benefit analysis of a PaaS versus Kubernetes

We're unable to provide you detailed financials for the Platform.sh PaaS of course, but relative to the above budget to build out Kubernetes, your cost-benefit planning assumptions to create the equivalent experience to a PaaS would be 10x the total build investment and 7x the ongoing annual operational cost.

The Kubernetes implementation described in the previous section will provide you:

- 30% of the value that a PaaS brings to the development workflow
- 40% of the value a PaaS brings to infrastructure management, ie. NoOps automation
- No uptime SLA for the live services

Kubernetes reference value vs PaaS: workflow, automation & live service



What you get from a PaaS

A PaaS is a proprietary service, but shouldn't expose anything proprietary to you. There should be zero vendor lock-in. Your developers should be able to describe - in the most succinct way possible - the requirements of their application, and the PaaS automatically deploys and manages those services. Should you decide to migrate away from your PaaS vendor, all you will be losing is the automation of the management layer; your code should run in precisely the same way on a highly-available PaaS using PostgreSQL or a MySQL cluster as it would with any other hosting vendor direct.

A PaaS should definitely give you an instant fast start for container management, developer workflow, support for a wide range of commonly required technologies, automated infrastructure management, automated deployments, automated maintenance, security updates, production service levels, seamless scaling, high availability, highly optimised resource consumption and cheaper compute/storage. Platform.sh basically transforms *"legacy applications"* into apps that have all the qualities of *12 factor apps**.

- The development community will be able to work in a totally different way, enabled by cheap development, test and staging environments-on-demand and in the cloud. Complete working copies of master/running sites, as many as required, and each one in less than 30 seconds (with Platform.sh). This means no more expensive dedicated test environments, zero set-up time for new environments, disposability of any environments, and super-fast testing and user acceptance of new features.
- Automated infrastructure management means no more DevOps. No more complicated processes to make things available to the development team for building and testing features.
- Failproof deployments straight into production (with Platform.sh).
- No-vendor lock-in to underlying tools sets or the IaaS itself. If you ever want to change your PaaS, you should be able to take your git repositories and YAML file configurations and reuse them elsewhere.
- A 99.99% uptime SLA in production, plus highly available cluster management, probably across a wide grid of resources. Plus the addition of resources with no live service interruption, for peak traffic upscaling as an example. (All with Platform.sh)
- Included infrastructure / application monitoring, debugging and analysis tools.
- 24*7 global support function, with experience of seeing and solving many similar implementation scenarios that look like your own (some PaaS vendors will have thousands of customers who they are learning from every day). So, whatever the uniqueness of your IT infrastructure, the problems you may be having are probably already known to them.
- Fast evolving roadmap of useful features.

In short, a PaaS will give you much higher productivity across the development teams, considerably lower costs and better efficiencies in operations, and much better economies for compute and storage resources, plus predictable pricing.

** 12 factor apps: https://12factor.net/*

Other savings and gains a PaaS will give you

When choosing a PaaS vendor, you should be asking them how they are able to evidence productivity improvements, better service management and lower costs, and if they are not yielding data driven evidence such as the following, buy from somebody that is:

METRIC CATEGORIES	EVIDENCE FROM CUSTOMERS
Fast development	
Set-up time improvement (new system)	Instant, months to days, 720x faster
Increased branching / better workflow	Worlds apart, 10-12x better
Developer productivity	20-40%, 100%, 300% more productive
Feature sign-off & UAT acceleration	500%, 700%, 14x faster
Fast deployment	
DevOps & ticket reduction	80-100% less SysAdmin and fraction of the tickets
Deployment time reduction	Faster & easier, never fails, 1500%
Deployment frequency improvement	1 a day to 10 per dev a day, 7x faster, every 2 hours
Live Service & Overall Costs	
Live performance	Awesome during peak, Phenomenal, Vast improvement
Interruptions and downtime in production	Zero, minimal
Overall cost reduction	60%, 4-5 FTE savings, 38%, 40%, 80%

What the PaaS customers should be saying

CANADIAN FOOTBALL LEAGUE

"Our focus is on creating business value, so it's important for my team to concentrate on writing code and not spend time worrying about deployment scripts, configuring and patching servers, and DevOps. With Platform.sh, putting new code live is as simple as a single command."

SULLY SYED
HEAD OF OPERATIONS

BRITISH COUNCIL

"Weekly changes used to take over 2.5 hours for all 130 sites around the world, but now takes less than 30 minutes. We can test and deploy emergency patches to all sites in less than 2 hours now, which was impossible before."

NICK MORGALLA
HEAD OF OPERATIONS

REISS

"Platform.sh was the best decision I ever made. I just can't imagine working without it now. The triple redundant architecture just works and the infrastructure is so fast and so performant."

PETER WARD
DEVELOPMENT MANAGER

TES GLOBAL

"We no longer pay for massive permanent resource allocations to meet unknown future peak traffic, because we know we can rely on a robust stack which scales seamlessly in minutes when needed."

AIDEN GREY
HEAD OF OPERATIONS

The Use Case for both PaaS and Kubernetes

Is a PaaS absolutely the right decision for your organisation.
Are you sure you're not missing a trick by passing up Kubernetes.

If your organisation believes a container based approach will benefit their implementation of various technology stacks, and that an automation framework is required to make this cost effective.

The following Use Cases are suited to a PaaS:

- When you are looking for a proven container based solution that will have a radical effect on the development teams' productivity with automated deployments to a Highly Available live service.
- If you want to run anything with a persistent workload, such as a database that require any level of consistency. To do this with Kubernetes you need to bolt various additional tools onto it.
- Single applications (OSS PHP Frameworks Drupal, Symfony, Ruby, Laravel, Magento etc.).
- Complex applications (Headless Drupal + Node'js).
- Micro-services applications with multiple persistent data backends.
- Enterprise NodeJS architectures.
- Enterprise with many technology stacks and an army of DevOps, Operations & Systems Administrators.
- Single-Tenant Software vendors launching cloud offerings.
- Multi-tenant SaaS vendors that want to simplify their operations and give their developer communities a fantastic new user experience.

...and these Use Cases are more suited to Kubernetes:

- If you need to run very large numbers of stateless clusters in a resilient fashion (this is the original Google Use Case fo Kubernetes)
- Large streaming workloads or Big Data workloads.
- The ability to run absolutely any service, built in any way you want, ie. the 1% of common requirements.
- You have a team of valuable DevOps and Systems Administration staff, that you would rather keep in house and in any case put to good use. Some European countries have onerous employment law that prevents the easy exit of staff, and other countries make it expensive in terms of redundancy payments.