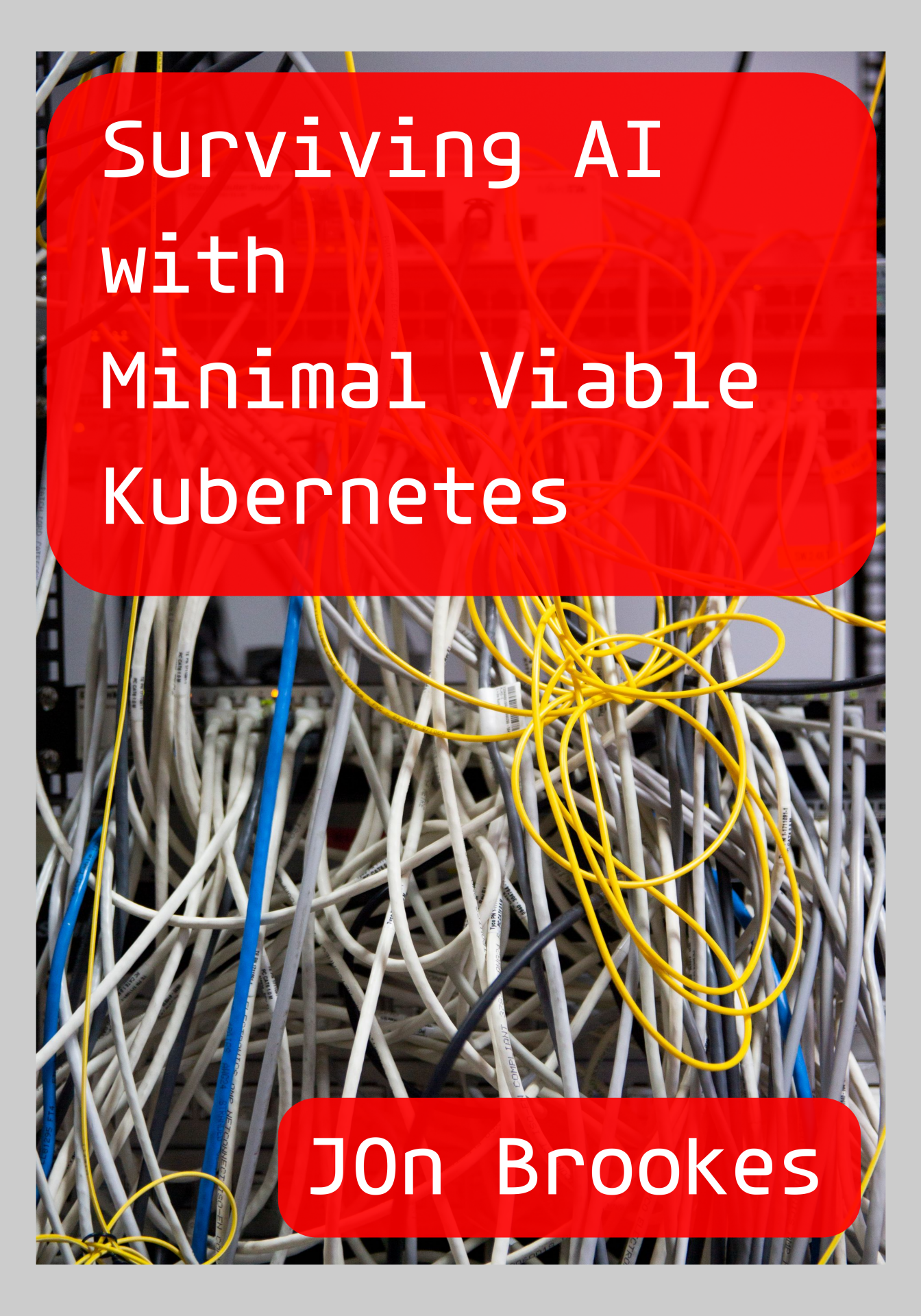




Surviving AI with Minimal Viable Kubernetes

Jon Brookes

Surviving AI with Minimal Viable Kubernetes

Jon Brookes

This book is available at

<https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>

This version was published on 2026-10-07



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Jon Brookes

Contents

- Who is this book for ? 1**
 - To Host or not to Host 1
 - Artificial Intelligence and our own Identity 2
- Surviving AI 4**
- The 3 pillars of K8s 5**
 - Ingress 5
 - Persistence 6
 - Etc 6
 - Minimal and Viable 7
- Up and Running 9**
 - Edge Single Node 9
 - 2 Node management plane / worker node 9
 - 3 Node Hyper converged management plane / worker nodes 10
 - The 4th Pillar of k8s 11
- Infrastructure is Ephemeral 12**
 - Routine Recovery and Rehearsal 12
- Deployments are Ephemeral 13**
 - Data Ownership 13
 - Scale to Zero 13
- MVK AI Stack 14**
 - Development 14
 - Production 14

Who is this book for ?

To Host or not to Host

This book is written by a person doing infrastructure for people that possibly don't do infrastructure, or perhaps somewhere in-between. It is also for those that already do infrastructure but want to simplify and reduce costs.

What do I mean by 'do infrastructure' ?

For over 20 years I've been working with technology and infrastructure, servers, software, deploying, supporting and developing platforms and applications.

I've done this in the era of the internet, and before the internet. I've worked in the era of SaaS and before that. I have early and lived experience of mainstream IT both before and after virtualization. And now, in the last few years, I've embraced AI to develop software and infrastructure and some of that software has become 'agentic' in nature.

Being a pragmatist, founder and solo developer / entrepreneur I have wanted to 'keep things stupid simple'. I don't want to pay a 3rd party to rent back to me all of my infrastructure if I can help it. This can lead to uncontrolled bills and financial ruin. Now, I'm just guessing here but I might suggest that you, the reader, may also be looking for similar. I hope that this book can help you to take back control of the cost of running your own software.

Where practical therefore, I choose to self-host software and services I use and sell. In order to do so, I lean quite heavily on my own experience over years of doing this for myself, clients, companies and enterprises. This book is my attempt to pass at least some of that on, so that you don't have to find things out the hard way.

I don't have a strong opinion on paying for SaaS. Where it makes economic sense to do so, SaaS is fine. But when your starting up a bootstrap or a small business and you're looking to cut costs, why pay enterprise rates, when you don't need to right now ?

How do I do this ? I do this by reducing each unit of infrastructure to run as containerized workloads and upon container runtimes that are running servers that I manage. Servers can be physical or virtual machines (VMs). In this way, costs can be more predictable. Then, by choosing the software and infrastructure carefully, I can keep overall costs and more importantly, cognitive load, in check.

Artificial Intelligence and our own Identity

AI has made us stop and look at who and what we are. Are we specialists, becoming highly focussed in one area of technology and business or are we generalists, working across multiple disciplines ?

If we are bent on being specialists, likely, this book is not for you.

The generalist, I believe, is what we need to become if we are to survive the post AI wave.

AI empowers us to do things that were out of reach in a persons lifetime of work. It gives us superpowers we did not have before. We can attempt tasks and work that we previously were not expert in but may have accrued some knowledge of over time.

Things that we know nothing about can be learnt quickly with AI. We can ask questions of the most patient one to one teaching assistant known to human kind.

The AI will not punish us for asking 'stupid questions'. It will entirely forget our past conversations if we ask it to. Our lack of knowledge is not a stigma, it has become a set of pins to knock down in a bowling alley.

The status quo that has existed for 100s of years has kept us in our places, being bakers, candlestick makers and the ones that put the nuts on the wheels in the Ford motor plant. Each has been given their own place, their own division of labour. We have kept in line and accepted our lot.

So long as we are allowed to use AI to 'fill in the gaps' and to increase our productivity in what we are already expert in, we can now stop outside of our previous, limited existence.

We can become hybrids. Polyglots, polymaths.

I have applied this thinking to software, infrastructure and infrastructure as code and specifically kubernetes. Kubernetes is perhaps not the best container

orchestration platform in the world but it has become the defacto. It is also complicated, very complicated. So I have looked at ways it can be de-mystified and unentangled from layers of over engineering.

If you're someone that wants to keep kubernetes complicated. so that that only a few can gain an understanding of it, likely, this is not for you.

If you are still reading this now, that is good and you are welcome. I don't expect you are exactly like me, and that is fine to just be curious. You perhaps a business owner or a start up or founder that wants to build and sell a thing but have been told Kubernetes is outside of your league.

The alternatives to Kubernetes are not as well supported and do not have as much adoption. And I don't believe that having to use self hosted servers in the way we did in the 90s is the way you have to run your business. Neither too do I believe you have to use plain old docker, compose and other proprietary, non kubernetes solutions as these you will likely need to *migrate from* at some point to a more competent kubernetes or managed solution in the future.

So why not? I believe k8s can be used by the individual, the solo entrepreneur, the tiger company startup of 3 to 5 people with some technical abilities under their belts but who cannot afford a team of devops and SRE engineers before they sell anything.

This book is how I have attempted to bridge the gap between knowing and not knowing how to orchestrate workloads on Kubernetes and knowing.

To do so, I created a 'Minimal Viable Kubernetes' (MVK).

This book is not intended to be an MVK manual. That manual is already at <https://mvk.headshed.dev> where you can implement it for yourself.

This is about how and why I got there and how you can use MVK and make it your own.

I hope that you will get something out of reading this so as to become your own boss and not just follow what others say you need to do and how you should run your own tech stacks.

There is a better way and I believe it is the MVK way.

Surviving AI

When AI happened, we were mostly all asleep to its emergence. It was only in 2022 that everyone became aware by the so-called main stream media.

In truth, we had all been using a form of prompt engineering with search engines before Chat-GPT. The AI search that has become the replacement to traditional internet search has simply added a conversation loop to what was already there.

It's infrastructure has been in development possibly since the start of the current century. AI was being developed well before that, as expert systems and chat bots.

Eliza was not even AI but users would talk to it as if it were human. Alan Turing suggested a test for computer intelligence and the Eliza program did a pretty good job of convincing users that they were happily chatting with a person. This was all without the use of a modern day neural network. It was just cleverly written code.

So we have been using artificial intelligence of some kind for years. We were not fully aware until Chat-GPT went mainstream in 2022 - when it was given a friendly face.

Being a good software engineer for over 20 years at least has often been about how quickly you could search for an answer. To do so became the verb 'to Google' for it. Remembering everything stopped being a requirement as it had been prior to the late 1990s when we had access to search engines.

What we have now is a 'better search' in which we can ask structured questions and get answers that we can argue with.

In building my minimal viable kubernetes, I have learnt to rely on chat based search and coding harnesses. I've been able to build faster.

Some say that AI leads to 'brain rot' where we lose the ability to think but I don't agree that is inevitable. I have not lost the knowledge I built over 20+ years of hacking infrastructure and the large language models have not made my knowledge and experience invalid. Brain rot I believe is a product of a lack of discipline, not a lack of knowledge.

The 3 pillars of K8s

Back in the mid 2010s I learnt kubernetes the hard way. I did follow <https://github.com/kelseyhightower/kubernetes-the-hard-way>. This literally is, building Kubernetes bit by bit, from binaries. You could say it is not for the faint of heart.

But this is how I learnt Linux, years before that, using <https://www.linuxfromscratch.org/> and <https://www.gentoo.org>. This is my way, not necessarily a recommendation but if your very serious about learning Linux and Kubernetes, it is one, possible path.

Another is <https://training.linuxfoundation.org/certification/certified-kubernetes-application-developer-ckad/> and I took this path, only to not sit the exams, being taken off to work on critically failing infrastructure at the time. I never took the time to re-book but I'm sort of glad I did not.

When I first worked on Kubernetes, it was from working on docker, Swarm and on self-hosted VMs and physical servers with self hosted hypervisors. One week after deploying to Kubernetes, I was on call for it 24x7. Not a pleasant experience, I can assure you.

In summary of my time running kubernetes, I found all roads that led to this devoid of 3 main areas of knowledge that I had to learn 'on the job' so to speak. Without these 3 things, I don't believe you can feasibly deploy containerized workloads. I call these 3 things, 'pillars'.

The 3 pillars of k8s that are glossed over by some training sources and managed hosting are

- Ingress
- Persistence
- Etcd

Ingress

In managed kubernetes, ingress, originally resulted in the automated creation of a load balancer for every service that was exposed. This was profitable for

the hyper scalers, not so for the SME running their first k8s cluster.

Over time, a single ingress became the norm, but led to each provider providing their own version of an ingress SaaS. So no one single style of ingress could work across all managed kubernetes providers.

Ingress has subsequently become a feature, left to each provider of managed kubernetes to do differently. Lifting and shifting from one managed platform to another is not going to be the same.

Self hosted k8s was even worse, as it has been left out in the cold so to speak but for third parties like [metalb](#).

In MVK, I use <https://k3s.io/> by Rancher and choose to install <https://traefik.io/traefik> using my own configuration, rather than the one provided by K3s

Persistence

Persistence, through persistent volumes and claims is an abstraction that should bring consistency across all kubernetes but it does not in my experience. Each managed kubernetes SaaS has its own storage layer, with their own mechanisms for managing them. Learning one is not learning them all.

In self-hosting, there is a similar state of affairs with different solutions offered by some proprietary, some open source. Complexity of implementation and management vary but few are trivial to setup and run. Each have pros and cons.

In MVK I chose to use <https://longhorn.io/> in 3 node+ hyper converged configurations.

Where single node K8s is appropriate - such as edge applications and bootstrap implementation, local storage comes as a part of K3s, out of the box.

Etcd

Etcd, at the heart of k8s. It is its state, memory, configuration database and more. It can run as a stand alone component to that of typical workloads. To run in High Availability (HA) requires 3 etcd instances. Self hosting, etcd can quickly become expensive and this is why many opt for managed kubernetes,

perceiving this as ‘the easier option’ if not to realize provider lock-in at a later date.

Additional to the subsequent hidden costs of managed kubernetes, the configuration of your infrastructure now becomes the concern of someone else. You lose the kind of access you would have in any kind of bare metal or self managed configuration. The specific management tooling of your etcd configuration in managed kubernetes becomes specific to each provider, each with different interfaces and learning curves.

Self hosting does not offer a ‘silver bullet’ or simple answer either.

Decision paralysis is inevitable in the face of the 3 pillars. A minimal viable kubernetes is my / our stab at ‘cutting through the noise’ in business speak.

In MVK, I chose to use k3s <https://github.com/k3s-io/kine> with a postgres back end in production, sqlite in development / CI/CD and native Etcd in 3 node+ hyper converged K3s clusters.

Minimal and Viable

With our 3 pillars in place, we have a firm footing for orchestrating container workloads within K8s, be it on VMs in the cloud, on-prem or on physical tin.

We have a reliable, consistent way of getting our users access to web services by a common ingress with Traefik at front and center.

Storage and persistence of containers is available as local drive space on single nodes, and can be shared across multiple nodes in hyper converged configurations.

By choosing to run K8s using K3s and on our own VMs or on physical hardware, we have a uniformity in terms of management that is cloud and even self hosting agnostic.

As the billing for virtual machines, at least in the current marketplace, remain relatively predictable, we have a way to curb unexpected bills.

I needed a small, fast, lean infrastructure to run my own business on. Managed k8s with one of the 3 main providers is an option but both I and my clients do not appreciate the lock-in that comes with managed kubernetes. To be able to move from one cloud provider to another and to our own on premises hosting should be possible and relatively friction free.

Without K8s specialist knowledge, small businesses, start ups and solo developers alike can quickly become locked out of all options but to settle monthly fees starting from 100s of dollars a month, even with zero workload.

I do not accept this as minimal or viable.

Neither too is having just self managed virtual machines, managed in the 'old ad-hoc shell command ways', as in 'servers as pets' with no containers or orchestration, acceptable today's threat models.

Having to run just docker on its own or with compose is a option but not that is easily scalable for business growth.

K8s out of the box though is complicated for the non specialist to get their heads around, making it the preserve of large enterprise organizations.

Minimal viable kubernetes is something I built to tackle the 3 pillars and the complexity problem to bring k8s to the masses.

It does this by choosing an opinionated design favouring simpler approaches over complexity and over engineering.

If I chat with AI about my intention to use kubernetes for smaller workloads, it may often push back on this approach, claiming that K8s alone is too complicated for the average user. It often states many of the reasons I set out to solve with MVK.

This should not really come as a surprise, as LLMs have been trained on the marketing and documentation which has been written by the hyper scalers - who want to sell us managed kubernetes.

Many decision makers are those with finance, rather than technical knowledge. Technical leaders are rarely rewarded as much and for 100s of years, engineers have been the 'doers' where as their masters have been the 'sayers' as it is they that oft hold the purse strings.

Surviving AI with Minimal Viable Kubernetes has been for me to draw on lived and worked experience. Sometimes, this can go against the grain of marketing that had become doctrine.

Up and Running

Up and running could likely be where the impatient folk like me jump to, skipping introductory sections. Welcome, either way.

“show me the code”

is my *own* normal reaction

So here is a quick ‘getting started’ in the most minimal viable way according to the ‘MVK way’.

Edge Single Node

MVK is built onto K3s, by Rancher.

Following the k3s documentation, the etcd layer of k3s uses a traditional database in place of etcd. It uses a sort of ‘shim’, an interface by which a database backed store can be read / written to by Kubernetes. The project it uses is also open source and called Kine, also by Rancher.

Kine can use SQLite, MySQL and Posgres databases. SQLite, their documents state, should not be used in production and only for testing and in CI/CD pipelines, so I opt here for Postgres, it being for me the most useful of the 2 remaining options.

I created a pipeline of my own to deploy a single node, edge style k3s using Kine, Postgres and k3s on a minimal virtual machine with 2 VCPU's and 4GB of memory. I add swap also for this minimal node, lest bad things happen and memory back pressure were to build. It at least gives me a few minutes to respond in such, before the node eats itself for want of memory.

Needless to say, this has to be carefully planned and monitored. But that is what MVK is all about.

T.B.C

2 Node management plane / worker node

Everything must scale

right ? At least that is what we are told by the internet and I am not going to say otherwise.

This is where docker alone, even with compose cannot truly meet this requirement aside that is of 'vertically', where a VM is given more VCPU's or memory or both. Downtime is typically be required and there are fixed upper limits to the size of VM you can scale to.

Kubernetes however can have 'worker nodes' added allowing 'horizontal scaling. Nodes can be 'marked' (given taints) to handle all or specific work loads. We can separate nodes by role, some to handle etcd state management, some to handle actual work loads.

A 2 node K8s is not really a cluster in any true sense of the word, as if either fail, typically, all bets are off. You may survive a management plane failure, if all workloads are on the remaining worker node but you have no means of managing anything.

Its important to understand that this is not a long term solution. Its just a stop gap for varying work loads. Longer term, we cand schedule for an afforded migration to true, 3 node+ cluster.

3 Node Hyper converged management plane / worker nodes

This is where some degree of high availability (HA) can be introduced in a minimal and viable way.

In an MVK 3 node configuration, each node is configured to act as both management and worker nodes.

The etcd kine shim is replaced by K8s etcd. So we have a 'real' k8s cluster spread across 3 VMs.

As we now have 3 nodes, we can use a storage layer that uses all 3. I may at some point like to use Ceph for this but to keep things simpler and to cut the costs of each node requiring 3 separate disks, I opted for Longhorn which

permits for the re-use of the VMs own root filesystem for storage. To save on cost and complexity, for now at least, I leave Ceph for another day and another version of the 3 node MVK.

T.B.C

The 4th Pillar of k8s

A fourth pillar, also I believe past by in many manuals are the apps that we want to run on kubernetes.

This is a consequence I believe, of the way kubernetes is deployed in typical SME's Enterprise and by hyper scalers.

As a Devops, SRE or Platform engineer, you rarely get chance to have an opinion on what applications to run. You get told to get on with it.

As a startup or founder, you are the one with an app already that you want to deploy and you make the decision.

The 4th pillar is therefore not only the apps you choose but the nature of deployment.

Each needs to follow a similar pattern.

Before any service goes into production I ask of myself:

1. is it backed up? is all user/customer data saved as backups
2. can I restore it? are the backups not made of toffee, can I really restore them and prove that they work?
3. is it monitored? are there alerts raised, sent and handled when services are lost or impaired? Are there meaningful service checks that can prove we have an available service for customers?

Out of this list, I have created a specification for MVK applications that subsequently have the following features

T.B.C

Infrastructure is Ephemeral

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Routine Recovery and Rehearsal

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Deployments are Ephemeral

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Data Ownership

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Scale to Zero

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

MVK AI Stack

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.

Production

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/surviving-ai-with-minimal-viable-kubernetes>.