Kubernetes: When and How to Apply It

BYTEBYTEGO
NOV 2, 2023 · PAID

Welcome back! In the first part of our Kubernetes deep dive, we covered the fundamentals - Kubernetes' architecture, key components like pods and controllers, and core capabilities like networking and storage.

Now, we'll dive into the practical side of Kubernetes. You'll learn when and how to apply Kubernetes based on your application needs and team skills. We'll explore advanced features, benefits and drawbacks, use cases where Kubernetes excels, and situations where it may be overkill.

By the end, you'll have a starting roadmap to putting Kubernetes into practice safely and successfully. Let's get started!

Kubernetes' Declarative Architecture

One of Kubernetes' key strengths is its declarative architecture. With declarative APIs, you specify the desired state of your application and Kubernetes handles reconciling the actual state to match it.

The Declarative Model

For example, to deploy an application, you would create a Deployment resource (discussed in the last issue) that declares details like:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: 'nginx:1.16'
          ports:
            - containerPort: 80
```

The Deployment resource declares the desired state:

- Use the nginx 1.16 image
- Run 3 replicas
- Match pods by app=my-app label

Kubernetes then handles all the underlying details of actually deploying and scaling your app based on your declared spec.

This is different from an imperative approach that would require step-by-step commands to deploy and update.

Custom Resource Definition

A key benefit of this architecture is extensibility. Kubernetes is designed to watch for new resource types and seamlessly handle them via declarative APIs. No modifications to Kubernetes itself are needed. Developers can create Custom Resource Definitions for new resource types that work just like built-ins.

Here is a simple example of Custom Resource Definition:

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: apps.mycompany.com
spec:
  group: mycompany.com
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: apps
    singular: app
    kind: App
```

This defines a new App resource under mycompany.com/v1. It could be used like:

```yaml
apiVersion: mycompany.com/v1
kind: App
metadata:
  name: my-app
spec:
  image: nginx:1.16
  replicas: 3
```

The declarative model enables powerful automation capabilities. Controllers can monitor resource specs and automatically adjust them as needed. For example, the HorizontalPodAutoscaler tracks metrics like CPU usage and scales Deployments up or down in response. The Cluster Autoscaler modulates node counts based on pod resource demands.

While Kubernetes does support imperative commands like kubectl run, these are less extensible and do not integrate with Kubernetes' automation capabilities as seamlessly. Using declarative APIs provides significant advantages in terms of extensibility, portability and self-service automation.

Advanced Built-in Resources

Kubernetes provides many built-in resources that leverage its declarative architecture to make managing applications easier. Some examples:

Ingress resources allow declarative configuration of external access to Kubernetes services. This leverages extensibility by introducing a custom resource to abstract the implementation details of exposing services. Different Ingress controllers can be implemented for various environments like Nginx, ALB, Traefik etc. This separation of concerns enables portability.

ConfigMaps provide a native Kubernetes way to inject configuration data into pods. ConfigMaps hold key-value data that can be mounted or set as environment variables. This allows separation of configuration from code/images. Pods directly consume ConfigMaps. This integrates configuration natively via Kubernetes' extensible API.

Role-Based Access Control (RBAC) introduces custom resources like Roles, RoleBindings and ClusterRoles. These combine to form access policies which enable granular permissions. RBAC deeply integrates declarative authorization into the Kubernetes API via admission control and enforcement.

Third-party Add-ons

In addition, a vibrant ecosystem of add-ons use Kubernetes APIs to extend functionality. Here are a few popular examples:

Helm introduces charts, which package YAML templates to declaratively manage complex, configurable applications. Charts can be hosted in repositories like package managers for easy sharing and installation. Helm effectively extends Kubernetes as an application platform.

Prometheus integrates via custom resources like ServiceMonitors for dynamic target discovery and autoscaling based on custom metrics. The Prometheus Operator and controllers leverage the extensible API to simplify monitoring, alerts and autoscaling.

Istio injects proxies for traffic control, observability and security using extensibility mechanisms like custom resources, controllers and admission webhooks. This layers on advanced features without changing code.

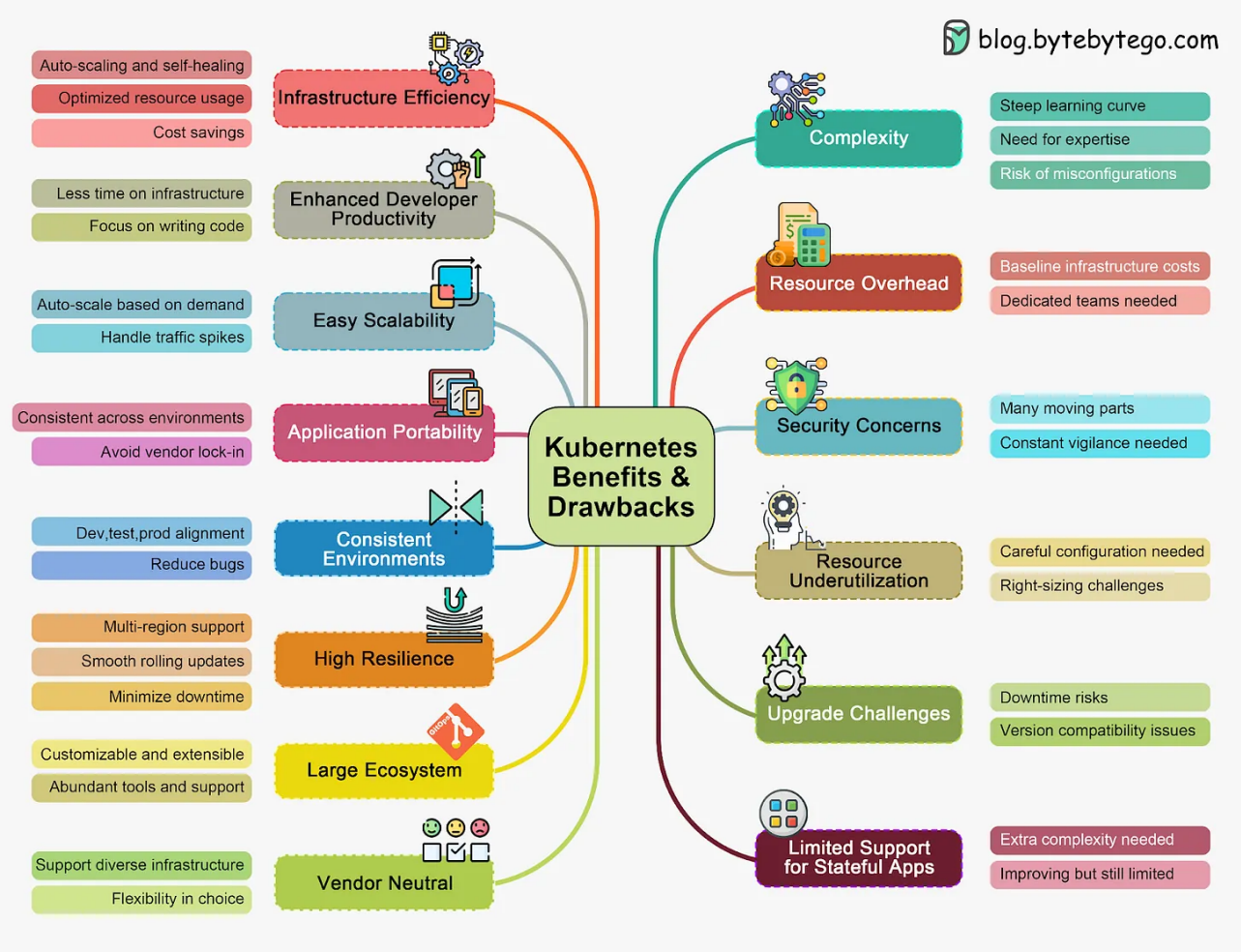
Argo CD utilizes custom resources, controllers, CRDs, webhooks and operators to enable GitOps workflows on Kubernetes. It models CD concepts through declarative APIs and reconciles state through automation.

Kubernetes lets you fully customize your system configuration. Everything is set up through Kubernetes resources and add-ons that you define. So you can shape the platform to your specific needs, rather than being constrained by predefined options.

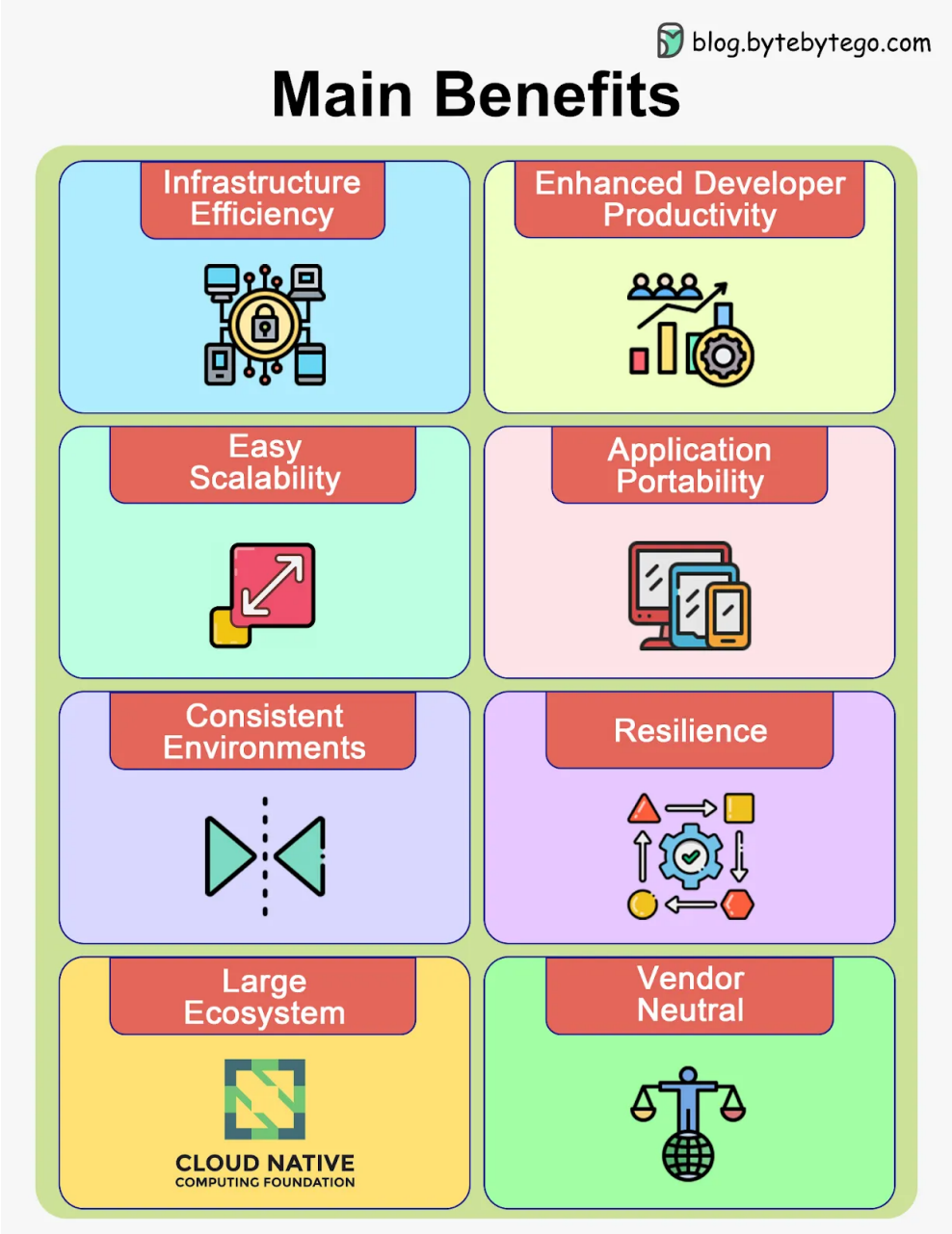
Kubernetes gives you common building blocks that you can combine in creative ways to meet your use cases. This open, programmable design means you can develop novel applications that even the original developers didn't think of.

Benefits and Drawbacks of Kubernetes

Kubernetes has become hugely popular due to the many benefits it offers, but like any technology, it also comes with some downsides that need careful evaluation. Below we dive into the key pros and cons of Kubernetes



The Main Benefits



Infrastructure Efficiency

One benefit of Kubernetes is how well it lets you manage resources on a large scale. By automatically scheduling containers across nodes, and features like auto-scaling and self-healing, Kubernetes makes sure resources are used optimally. It adjusts smoothly based on demand, starting up or shutting down resources as needed. This not only saves money from better resource use, but also guarantees applications have the right resources to run smoothly.

Enhanced Developer Productivity

By handling many infrastructure tasks like scaling and deployments automatically, Kubernetes frees developers from having to spend as much time on those tasks. This

lets developers focus their time and energy on writing application code rather than worry about infrastructure details. This boosts developer productivity once Kubernetes is set up.

Easy Scalability

A key advantage of Kubernetes is how easily it allows applications to scale up and down based on real-time demand. This makes sure resources are used efficiently while performing well during traffic spikes. The auto-scaling is especially useful for applications with changing workloads.

Application Portability

Kubernetes provides a consistent deployment experience whether you run applications on-premises, in the public cloud, or hybrid. This makes it easy to move applications between environments and reduces dependency on any single cloud provider.

Consistent Environments

Kubernetes provides the same standardized environment for your code in development, testing and production. This reduces bugs from inconsistencies between environments. Developers and QA engineers get a reliable testing environment that closely matches production.

Resilience

Kubernetes provides capabilities that allow developers to build highly resilient applications, such as automatic scaling, rolling updates, and redundancy across regions. However, these resilience capabilities need to be purposefully implemented using Kubernetes' flexible architecture.

Large Ecosystem

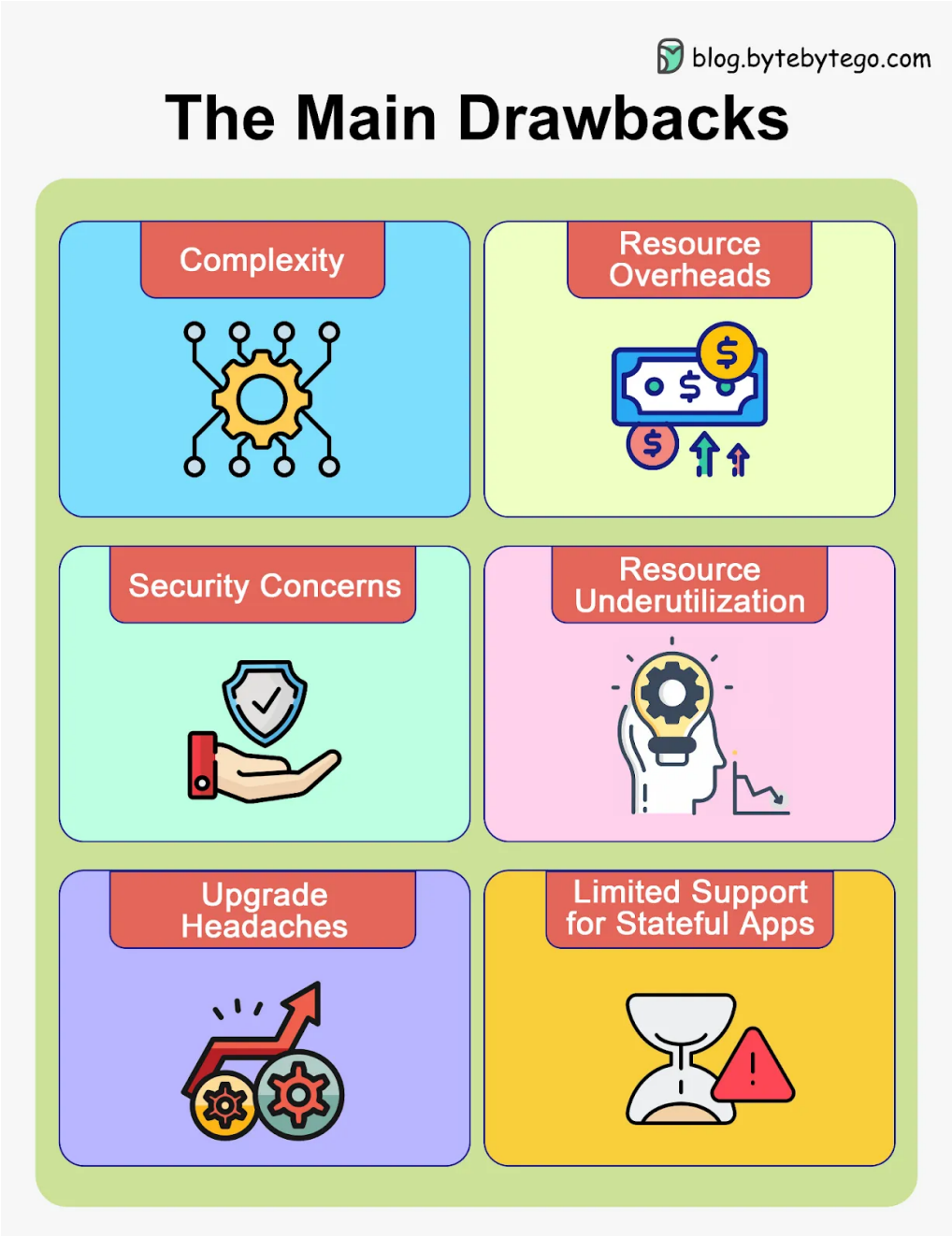
The active open source community around Kubernetes has created many tools, plugins, extensions and resources that make it customizable for diverse uses. Help and support are readily available.

Vendor Neutral

With support for diverse infrastructure both on-premises and across public clouds, Kubernetes prevents locking into a single vendor. You have flexibility to choose

suitable infrastructure for your needs.

The Main Drawbacks



Complexity

The biggest downside of Kubernetes is its complexity.

The extensive capabilities of Kubernetes also make it complex, especially for production-grade deployments with many moving parts working in concert. It demands significant expertise to set up and manage properly.

Ongoing management and troubleshooting also requires specialized engineering skills. Misconfigurations can easily take down applications deployed on Kubernetes.

For smaller teams without deep DevOps skills, this complexity can outweigh the benefits. The learning curve is steep, especially for those new to large-scale distributed systems.

Resource Overheads

Running Kubernetes comes with overhead resource costs. A Kubernetes control plane requires a certain baseline level of resources.

For smaller applications or organizations just starting out, these overhead costs may not justify the benefits. The resources required to run Kubernetes would be excessive.

The human resources required to properly operate Kubernetes also imposes overhead costs. At high scale, dedicated teams are needed to manage Kubernetes infrastructure full-time.

For many smaller teams, these operational costs are difficult to justify. The break-even point at which Kubernetes efficiency pays off is higher.

Security Concerns

Securing a Kubernetes environment is challenging given the platform's many configurable parts. It requires solid expertise and constant vigilance to lock down properly.

Resource Underutilization

While Kubernetes aims to optimize resource usage, improper configuration can also lead to overprovisioning and waste. Right-sizing cluster resources based on actual utilization is critical.

Upgrade Headaches

Keeping Kubernetes clusters up-to-date requires careful planning and execution to minimize application downtime and maintain compatibility during upgrades.

Limited Support for Stateful Apps

While optimized for stateless workloads, Kubernetes does provide capabilities to support stateful applications through features like StatefulSets, persistent volumes, and affinity/anti-affinity. While support for stateful apps like databases is improving, running them on Kubernetes still involves additional complexity.

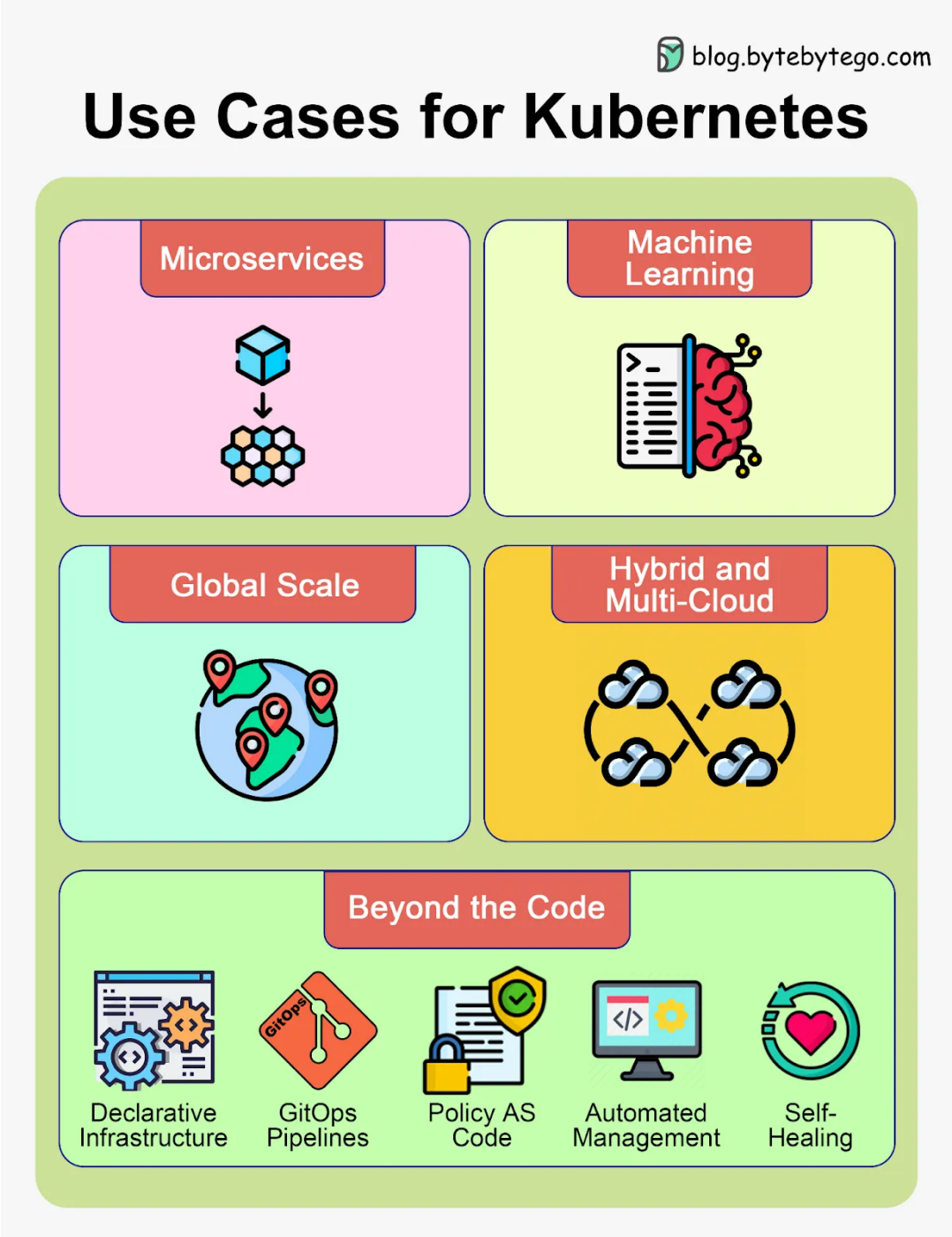
Our Take

Kubernetes offers immense benefits but also comes with drawbacks. It excels for large-scale, distributed applications that require portability across environments, high availability, and operational efficiency. But it also introduces complexity and resource overheads.

As with all engineering decisions, tradeoffs must be evaluated based on application needs and team constraints. Both the technical pros and cons along with business considerations like costs and capabilities drive the appropriate decision.

Use Cases for Kubernetes

Kubernetes shines for large-scale, complex applications. But is it a good fit for every workload? Let's examine key use cases where Kubernetes excels and also where it may be overkill.



Microservices

Microservices architectures break applications into small, independent services. This matches well with Kubernetes.
Each microservice can be packaged as its own container image. Kubernetes can then deploy and manage each one separately.

As traffic patterns change, Kubernetes can smoothly grow or shrink the number of copies of each service. New versions can roll out through staged updates without downtime. Kubernetes handles these tricky coordination jobs automatically.

For example, Spotify faced several challenges in managing its microservices architecture:

- Hundreds of separate microservices built by independent teams
- Frequent new service releases
- Unpredictable traffic and spikes in usage
- Massive scale - over 1600 services and growing rapidly

Before Kubernetes, Spotify used homemade tools to manage virtual machines. This created bottlenecks that slowed innovation. Engineers lacked self-service access to resources.

To address these challenges, Spotify adopted Kubernetes for:

- Packaged each microservice as a container image for independent deployment
- Leveraged Kubernetes automation for rapid deployments from dev to prod
- Dynamic scaling of pods to handle bursts in traffic
- Rolling updates to frequently release new features
- Self-service access for engineers to resources without bottlenecks

The impact of using Kubernetes included:

- Increased developer productivity and happiness
- Ability to scale to over 1000 microservices and thousands of instances
- Rapid innovation velocity - over 100 new microservices per month
- Quick adaptation to changes in usage patterns
- Avoidance of bottlenecks for developers releasing updates

Machine Learning

Kubernetes helps optimize machine learning workloads through features like GPU scheduling, priority classes, and node affinity. This improves resource allocation between training and inference jobs, boosting ML workload performance.

OpenAI was launching several groundbreaking AI products like GPT and DALL-E and knew they needed to overhaul their infrastructure to support massive growth in

traffic and machine learning workloads. Their then-existing infrastructure was on Amazon Web Services (AWS). But as their computational demands outgrew AWS' capabilities to cost-effectively scale, they looked to modernize their architecture.

They decided to adopt Kubernetes to automate and optimize their environment. Kubernetes offered a cloud-agnostic platform to run OpenAI's workloads. After adopting Kubernetes, OpenAI could develop and deploy applications independently of the underlying cloud provider.

A major motivation was improving support for their AI research, which involves computationally-intensive training of large neural network models on GPUs. Their old infrastructure made GPU resources hard to allocate efficiently. Kubernetes allowed fine-grained control over GPUs using pod requests and limits. Engineers could dynamically provision GPU clusters tailored to each training job's needs. This improved overall GPU utilization from 30% to over 80%, critical for reducing training costs.

Kubernetes also enabled seamless autoscaling of training jobs. Researchers could launch hundreds of trials in parallel, iterating rapidly to find the best model architectures and hyperparameters.

For production deployment, Kubernetes minimized the latency of serving AI models to end users. Models could be made available in just 2 days instead of 2 weeks previously. Real-time autoscaling ensured users had low-latency access to compute resources for inference.

More broadly, Kubernetes accelerated OpenAI's development workflows. The API enabled programmatic management of infrastructure, allowing engineers to deploy code multiple times per day compared to weekly in the past. New services could be launched in days without performance hits - for example, 16 major services deployed in the first month after migration.

Kubernetes reduced costs by 40% by eliminating the need to provision fixed servers. Resources could scale up and down based on live traffic patterns. Outages were also reduced by 60% thanks to health checks and self-healing.

OpenAI started by using Google's GKE service to quickly get started with Kubernetes. When Google Cloud infrastructure costs became untenable, Kubernetes

enabled OpenAI to seamlessly migrate to Microsoft Azure within weeks.

Migrating to Kubernetes gave OpenAI the automation, scalability, reliability and multi-cloud portability required to rapidly experiment with emerging AI models while efficiently serving them at global scale. The launch of ChatGPT highlights the transformative impact this cloud-native infrastructure had on enabling their research and product innovation.

Global Scale

Operating applications globally that need to serve users across the world with low latency is incredibly complex. Kubernetes provides valuable primitives for distributed coordination between clusters spanning multiple regions to help manage this complexity.

Global traffic management patterns like geo-DNS, anycast IPs, and global load balancing further help reduce latency for users worldwide. Sophisticated Kubernetes features like multiple schedulers and federation simplify operating massive global Kubernetes footprints across regions and zones.

Multiple Kubernetes scheduler instances can be run across regions. This allows scheduling decisions to be made closer to where pods run, improving performance. Federation abstracts away cluster differences so users can operate on familiar Kubernetes API objects across distant clusters.

For example, when Pokemon Go launched, it leveraged Kubernetes on Google Container Engine (GKE) to scale its cluster globally on demand. This allowed Pokemon Go to orchestrate containers at a global scale to serve users where they were.

Kubernetes gave the team freedom to focus on rapidly deploying features to delight Pokemon Go's global fan base. The initial Kubernetes deployment was the largest on GKE at the time.

When Pokemon Go launched in new countries, it got way more players than expected. Pokemon Go prepared for a huge flood of new users, but the number of players vastly exceeded even their worst predictions. To handle all the extra players, they added tens of thousands more computing cores to expand Pokemon Go's

Kubernetes cluster on GKE. This supported over 50 times the number of players they had originally planned for.

Kubernetes features like multiple schedulers and federation were crucial for Pokemon Go to manage explosive demand globally. As millions of new players signed up daily across the world, these capabilities kept the game smooth and responsive.

The success of Pokemon Go shows how companies can leverage Kubernetes globally, using patterns like geo-DNS, anycast IPs, and global load balancing to reduce latency. This unlocks the next generation of global-scale applications.

Hybrid and Multi-Cloud

Modern enterprises increasingly operate complex hybrid environments spanning multiple private data centers across regions and public clouds like AWS, GCP, and Azure.

Kubernetes provides a consistent control plane to unify workflow across this heterogeneous, distributed footprint. The same Kubernetes API can be used for deploying containers to a private data center in New York and a public cloud instance in Singapore.

This cloud-agnostic portability allows developers to focus on building robust applications without worrying about underlying infrastructure. Ops teams gain flexibility to manage hybrid infrastructure as code.

With unified abstractions for networking, storage, security, and other resources, Kubernetes enables seamless bridging of on-prem and public cloud environments. Services built as distributed microservices can span data centers and clouds.

Leading financial data firm Bloomberg runs hundreds of Kubernetes clusters with thousands of nodes each to orchestrate services across its massive private network and public cloud.

Bloomberg's developers leverage Kubernetes to deploy apps and data pipelines swiftly from their terminal in London to on-prem infrastructure in Tokyo. Unified Kubernetes workflow makes their globally distributed private cloud feel as agile as public clouds.

Kubernetes provides essential consistency for hybrid cloud. It allows enterprises to standardize deployment on a unified control plane while preventing cloud vendor lock-in. Workloads can be migrated between environments or balance workloads based on cost and performance.

As Bloomberg's experience shows, Kubernetes unlocks developer productivity and operational flexibility to bridge on-prem and cloud. Hybrid Kubernetes unifies workflow while avoiding sacrificing the unique benefits of public and private environments.

Beyond the Code

Kubernetes provides a way to manage infrastructure that goes far beyond just running application code. With Kubernetes, teams can take an "infrastructure-as-code" approach to automate and optimize their entire environment.

Some key ways Kubernetes enables infrastructure-as-code:

- Declarative Infrastructure - Kubernetes configurations allow you to declare the desired state of infrastructure like networks, storage, security policies, etc. Kubernetes works to match the actual state to the desired state. This is more efficient than procedural, scripted infrastructure changes.
- GitOps Pipelines - Kubernetes has support for GitOps workflows which rely on Git as a single source of truth for both code and infrastructure. Infrastructure changes are version controlled alongside code changes. A GitOps agent like ArgoCD syncs Kubernetes cluster state to the Git repository.
- Policy as Code - Security and governance policies like RBAC and network policies can be defined as code in Kubernetes. This allows consistent policy across environments. Policies can be version controlled and tested like code.
- Automated Management - Kubernetes controllers work to automatically roll out changes, detect issues, restart components and more without human intervention. This automates routine management tasks.
- Improved Monitoring - Kubernetes has native support for monitoring pipelines, logging and metrics. This provides observability into infrastructure as well as applications.

- Self-Healing Capabilities - Kubernetes can self-heal certain failures, restarting containers, replacing nodes, rescheduling workloads and more. This increases resilience.

Together these capabilities allow teams to apply software development best practices like version control, testing and automation to infrastructure management. Kubernetes enables infrastructure to be managed fully as code.

When Kubernetes May Not Be the Best Fit

Kubernetes has emerged as the leading platform for deploying and managing containerized applications. However, despite its popularity, Kubernetes may not always be the ideal solution. It's important to evaluate your specific use case to determine if Kubernetes is the right fit or if alternatives like Docker Swarm, autoscaling virtual machine clusters or serverless architectures are better suited.

Small-Scale Deployments

For smaller workloads or limited resources, the overhead of running Kubernetes may outweigh the benefits. The complexity and resource requirements of setting up and operating a Kubernetes cluster introduces considerable operational overhead compared to simpler solutions. This makes it less ideal for small companies, projects with constrained resources, or applications with minimal scale.

Simple Applications

Applications that are monolithic or have simple architectures often don't require the full capabilities of Kubernetes. For apps that don't demand distributed components or microservices, introducing Kubernetes may add pointless complexity. Traditional deployment approaches like a simple cluster of VMs behind an application load balancer may be more appropriate for simple, lightweight apps.

Rapid Development and Prototyping

When rapid iteration and prototyping are the priorities, Kubernetes can slow down development velocity. The learning curve plus the effort required to configure and deploy clusters makes Kubernetes less suitable for prototyping new ideas quickly. Lightweight platforms like AWS Fargate and Google App Engine allow developers to build and test prototypes faster by reducing operational concerns. The simplicity and automatic management of these platforms enables quick experimentation without the complexity of configuring and managing a full Kubernetes cluster. For rapidly developing and assessing new concepts, these platforms allow teams to focus on iterating rather than Kubernetes cluster administration.

Lack of Expertise

Kubernetes' complexity means a lack of proficiency can lead to misconfigurations, security issues, and inefficient resource utilization. Before adopting Kubernetes, teams should honestly assess their capacity and willingness to learn. For teams without container orchestration experience, managed Kubernetes offerings can help lower the barriers to entry. Alternatively, fully managed "serverless" container platforms like AWS Fargate and Google Cloud Run may be easier starting points than managed Kubernetes offerings, as they abstract away more operational complexity.

Cost Considerations

The resource requirements of Kubernetes control plane components and worker nodes often entail increased costs over simpler hosting solutions. The scaling and automation strengths of Kubernetes may justify these expenses for large workloads, but teams should evaluate cost/benefit tradeoffs.

Legacy System Integration

Older, monolithic applications can pose challenges for adoption of Kubernetes and containers. Significant refactoring may be required to containerize legacy apps and

decompose them into microservices. For some legacy systems, this level of change may be infeasible.

Our Take

Evaluate workload needs and team capabilities before adopting Kubernetes. For large-scale, dynamic workloads under expert DevOps operation, Kubernetes delivers immense value. But it also introduces complexity and resource costs - which can worsen outcomes for small, simple or static workloads.

Right size your technology. Let your workload drive and team constraints steer your Kubernetes decision.

Conclusion

Kubernetes has quickly become the standard for container orchestration. But it is not a panacea.

The key to success with Kubernetes is to understand its strengths and apply it to suitable workloads. Used properly, it can simplify operations and accelerate delivery for complex, distributed applications. But misapplied, its overhead can impede productivity.

The most common success factors for Kubernetes adoption are:

- Applications composed of many microservices with dynamic resource demands
- Need for portability across on-prem, cloud, and hybrid environments
- Infrastructure that must handle frequent changes like autoscaling and rolling updates
- Teams with strong DevOps skills to handle Kubernetes complexity

Organizations that meet these criteria are well-positioned to benefit. Kubernetes will provide material improvements in productivity, reliability, and efficiency.

The economics also shift favorably at scale. The fixed overhead costs of Kubernetes get amortized over many nodes and applications.

On the other hand, use caution with Kubernetes if:

- Your applications are simple or have static resource requirements
- Environment portability is not a priority
- Your team lacks deep operational experience with distributed systems

In these cases, the orchestration complexity may hinder more than help.

Of course, few technology decisions are black and white. Evaluate the tradeoffs closely for your needs. You may find opportunities to start small with Kubernetes and expand its scope over time. Used judiciously, Kubernetes provides a powerful lever for scaling application delivery. Learn it, apply it wisely, and unlock its benefits safely. But don't force-fit it everywhere out of hype.

This concludes our deep dive into Kubernetes - where it excels, where it falls short, and how to make it work for your applications.

2 Comments