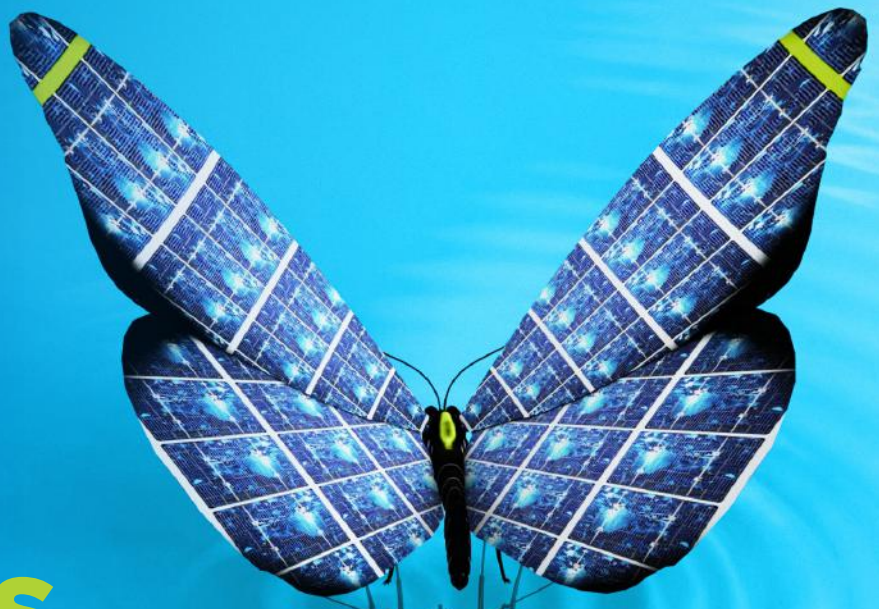
NUTANIX



7 WAYS TO SIMPLIFY

KUBERNETES LIFECYCLE MANAGEMENT

7 WAYS TO SIMPLIFY

KUBERNETES LIFECYCLE MANAGEMENT

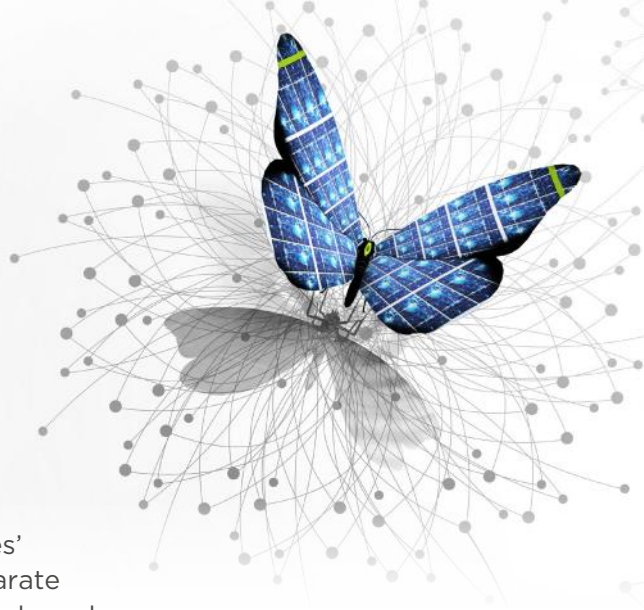
The rapid enterprise adoption of cloud native technologies puts Kubernetes at the center of almost every major digital transformation project. Driven by an expansive open source community, Kubernetes and its growing ecosystem are evolving at breakneck pace, with substantial new functionality being released on a quarterly cadence. Public cloud providers offering up a basic Kubernetes platform-as-a-service make it deceptively simple to operationalize and use the latest version. However, enterprises seeking to build their own on-premises Kubernetes environments face complex technical challenges across cluster and application lifecycles. Without the right tools and approach, IT Operations risks becoming a major roadblock to developer productivity and compromising the overall health of their on-premises Kubernetes environments.

This report outlines 7 of the most challenging aspects of Kubernetes that IT operations teams face and offers recommendations on how to simplify them through the application of intelligent automation and dedicated solutions.

01

CLUSTER UPGRADES

KUBERNETES AND HOST OS



With major Kubernetes releases rolling out quarterly, IT Ops finds itself on the hook to quickly upgrade Kubernetes across all cluster nodes, especially in the event that a security vulnerability has been identified and patched. By default, upgrades are completely manual, requiring SSH'ing to each node and running the necessary commands which vary based on Kubernetes version and host OS.

Kubernetes Master Node Upgrade Process

1. Drain the first master node (which incurs downtime, unless two or more master Kubernetes nodes are running)
2. Upgrade the cluster orchestrator / infra piece (typically kubeadm, but there are others) on that master node
3. Upgrade the master control plane
4. Upgrade the master kubelet and kubectl
5. Uncordon the upgraded master node
6. Repeat steps 1 through 5 for each of the remaining master nodes

Kubernetes Worker Node Upgrade Process

7. Drain the first worker node (hopefully there is enough capacity to avoid application downtime)
8. Upgrade cluster infra (kubeadm) on the worker node
9. Upgrade kubelet and kubectl on the worker node
10. Uncordon the worker node
11. Repeat steps 7 through 10 for each of the remaining of the worker nodes

etcd Upgrades

Upgrading etcd (Kubernetes' primary datastore) is a separate process. The exact commands and procedures will vary based on the underlying OS and cluster creation method, and Kubernetes version. Detailed information on how Kubernetes can be upgraded is available **here**.

Host OS Upgrades

Upgrading the Host OS is a similar process to upgrading the Kubernetes version, where each node is drained one at a time, upgraded (sudo yum upgrade or sudo apt {update;upgrade}), rebooted, and then uncordoned.

Automation around upgrades dramatically simplifies these processes, freeing up IT operations from a lot of messy work. A well-designed Kubernetes management system should enable push-button upgrades with zero disruption to running applications.

02

PERSISTENT STORAGE

The ephemeral nature of containers makes it difficult to persist data, and stateful applications depend on provisioned storage remaining connected to the Pods that host them. With an increasing number of stateful applications being refactored or implemented using cloud native principles, storage has become a widespread challenge.

While Kubernetes now offers a standard mechanism for exposing arbitrary block and file storage systems to containerized workloads with its Container Storage Interface (CSI), there is still a lot of key decision-making and management overhead related to storage on the part of IT operations and application owners.

Key considerations are the type of storage to be used, how it will be made accessible to Kubernetes clusters, and how it will be provisioned and used by applications.

What should users ultimately look for in a Kubernetes storage solution?

Support for major storage classes

It is highly recommended to use a storage solution offering support for file, block, and object storage classes. Different applications value different mediums; some, such as databases, require the fast performance of block storage, some may depend on the ability for multiple Pods to access the same storage (file storage with read-write-many), and some may require simple configuration and enormous scale (object storage). Often, a single application will use multiple storage types. JFrog Artifactory, for instance, works really well by placing incoming writes on block-based persistent storage volumes for optimal performance, and then tiering these artifacts to distributed object storage for optimal scale.

Automation

Persistent storage solutions for Kubernetes should also automate the installation of CSI drivers on every Kubernetes cluster, along with the creation of a default storage class. In the event a Pod restarts, abstracted storage should be automatically re-mounted. Lastly, best-in-class Kubernetes storage solutions should support dynamic volumes, which allow administrators to create high-level storage classes and enable developers to create Persistent Volume Claims (PVCs) where storage is automatically provisioned from a matching storage class.

03

SECRETS MANAGEMENT

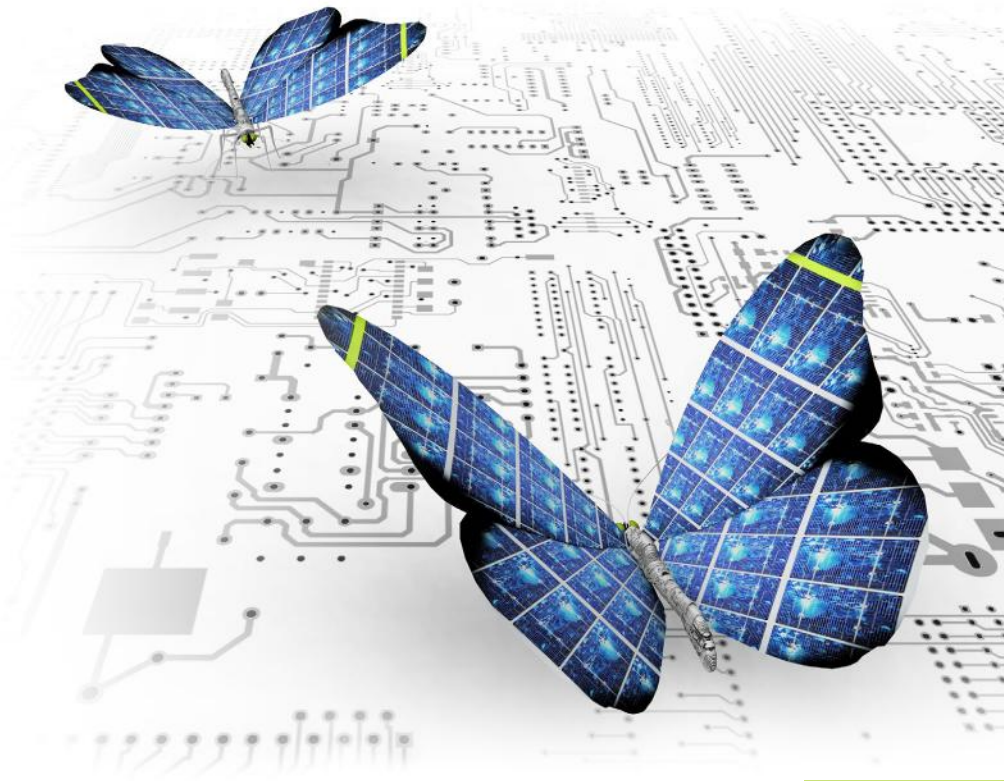
Security is undoubtedly the biggest challenge for enterprises who build and manage their own Kubernetes environments, and a key area that burdens both IT operations and security teams alike is the management of sensitive data in Kubernetes environments, known as 'secrets'. Secrets are Kubernetes objects used to store SSH keys, tokens, passwords, etc. that are required when containerized applications need to interface with other systems.

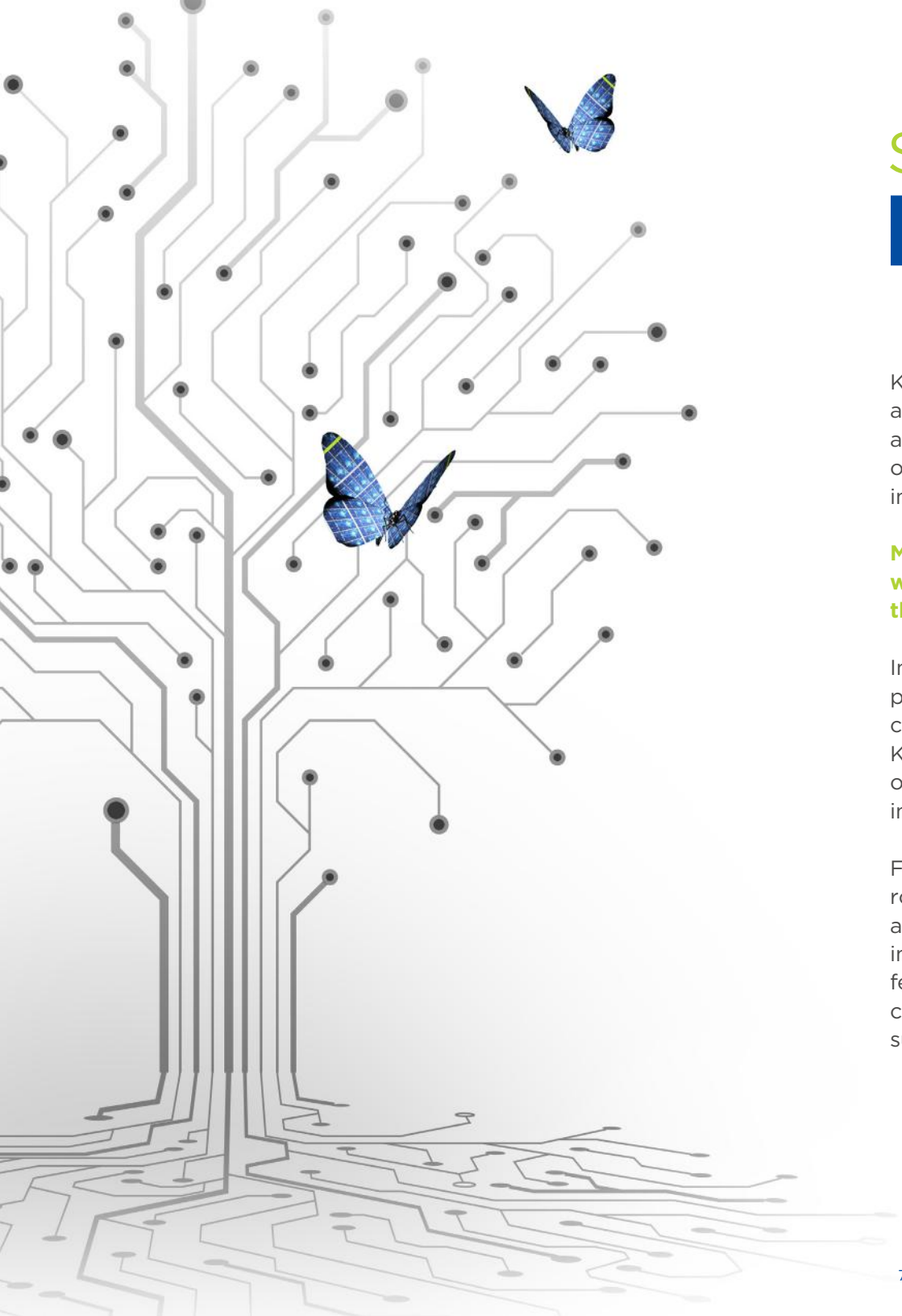
By default, secrets can be accessed using kubectl or via the Kubernetes API. The output is encoded, so they're not in clear text. However, this encoding does little to protect the secret as it can be easily decoded. Secrets are stored in plaintext on the etcd server and unless the cluster is configured to encrypt communication via TLS, these secrets also go over the wire in clear text.

Kubernetes provides some basic security capabilities around secrets (encryption, policies, and whitelist access) but they require enforcement. In addition to safeguarding secrets data throughout its lifecycle, administrators also need to focus on ensuring that any changes to secrets don't break applications in production.

Kubernetes administrators should seek out a dedicated secrets management solution that works on individual containers. Furthermore, change management capabilities are critical (automatically pushing changed secrets to the application containers that rely on them).

The major public cloud providers each provide their own protection and management mechanisms for secrets, but an open source tool such as Hashicorp Vault might be a better choice for users seeking to avoid lock-in.





SERVICE DISCOVERY

Kubernetes assigns Pods their own IP addresses, as well as a single DNS name for a set of Pods, and can load-balance across them. However, because a Pod can be scheduled on one cluster node and later be moved to another, any internal IPs that this Pod is assigned can change over time.

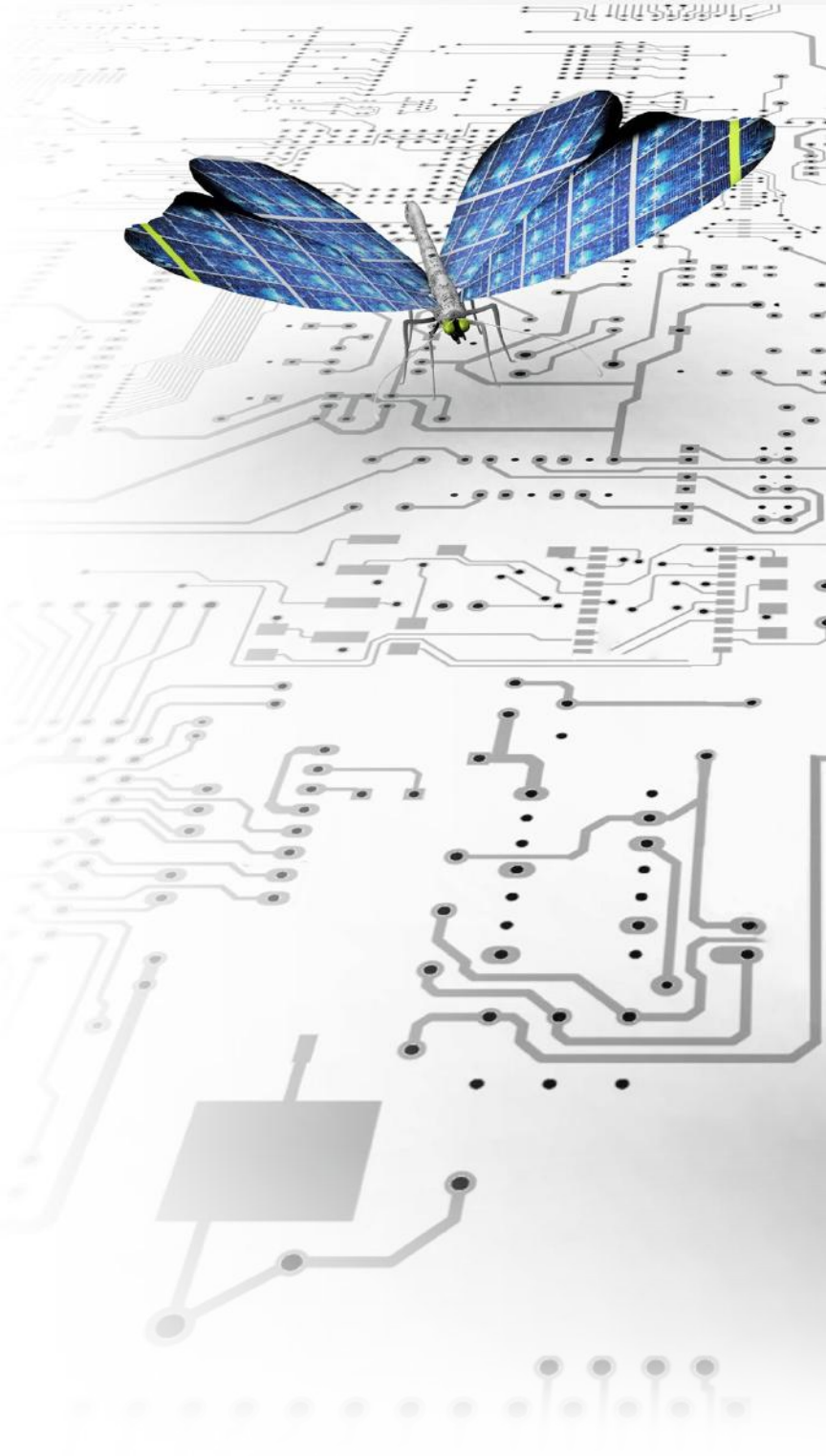
Making Pods reachable to external networks or clusters without relying on any internal IPs is a technical challenge that ultimately requires another layer of abstraction.

In this case, a load balancer is an invaluable solution that provides each Pod a unique IP that is accessible outside the cluster. Load balancing functionality is not natively part of Kubernetes; it relies on the infrastructure provider, or on an open-source tool like MetalLB. The container network interface (CNI) can also enable routing to individual pods.

For business-critical applications which require high robustness, a Kubernetes Ingress is recommended. Ingress allows for name-based virtual hosting and SSL termination, in addition to load balancing. While Ingress is the most feature-rich of these capabilities, it is also the most complicated, and relies on a third party Ingress controller such as Nginx, Traefik, or Istio.

05

APPLICATION MANAGEMENT



A single Kubernetes application may consist of several services spanning dozens of containers, and configured with Persistent Volumes, Secrets, and StatefulSets.

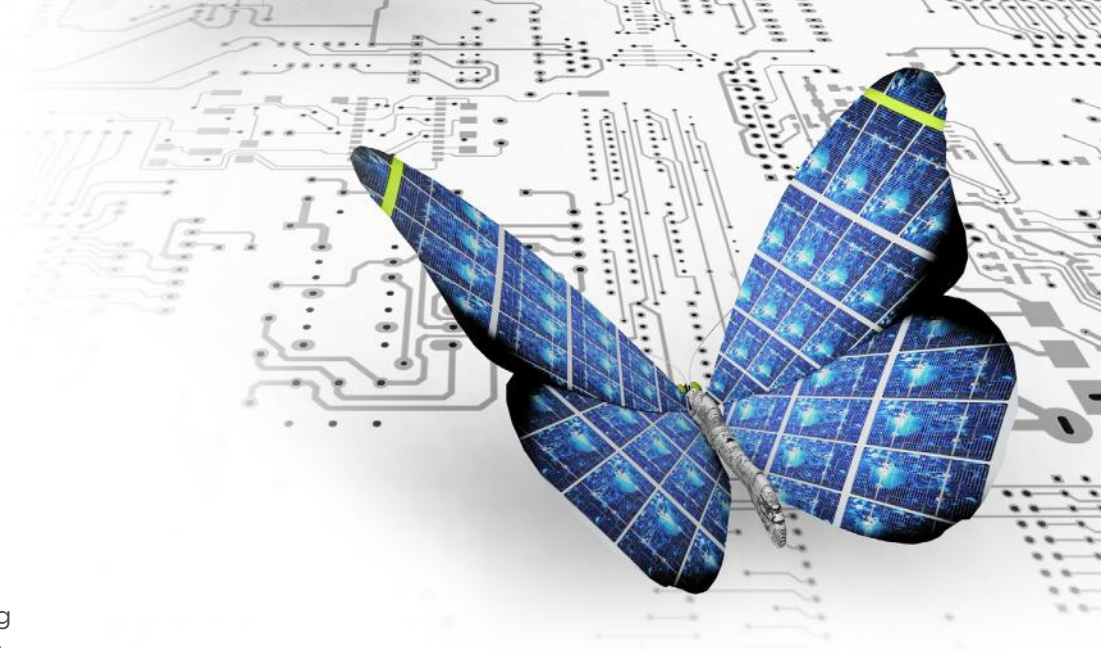
While grouping each application into a unique namespace can help simplify and organize your Kubernetes cluster, managing dozens or hundreds of Kubernetes-based applications this way does not scale, and will quickly overwhelm teams managing the environment. Here, a third party tool which manages deploying, modifying, tracking changes, upgrading, and reclaiming entire applications is essential.

Helm is one of the most popular tools that provides this functionality, however, it also introduces a new set of challenges around tracking application configuration in source control management while locking down access to prevent untracked changes. Kubernetes Operators were created to solve these challenges, and are recommended for production workloads. Building an Operator may take weeks to months, and involve writing thousands of lines of code, but the benefits are significant. For example, operators allow for IT team members who are not experts in a particular application to be able to effectively manage it, rather than having the application's developers do all of the management work. Using operators, software developers can write code once on how an application should be upgraded, and then the Kubernetes administrator can easily initiate the upgrade whenever it's needed.

06

CLUSTER HEALTH

MONITORING



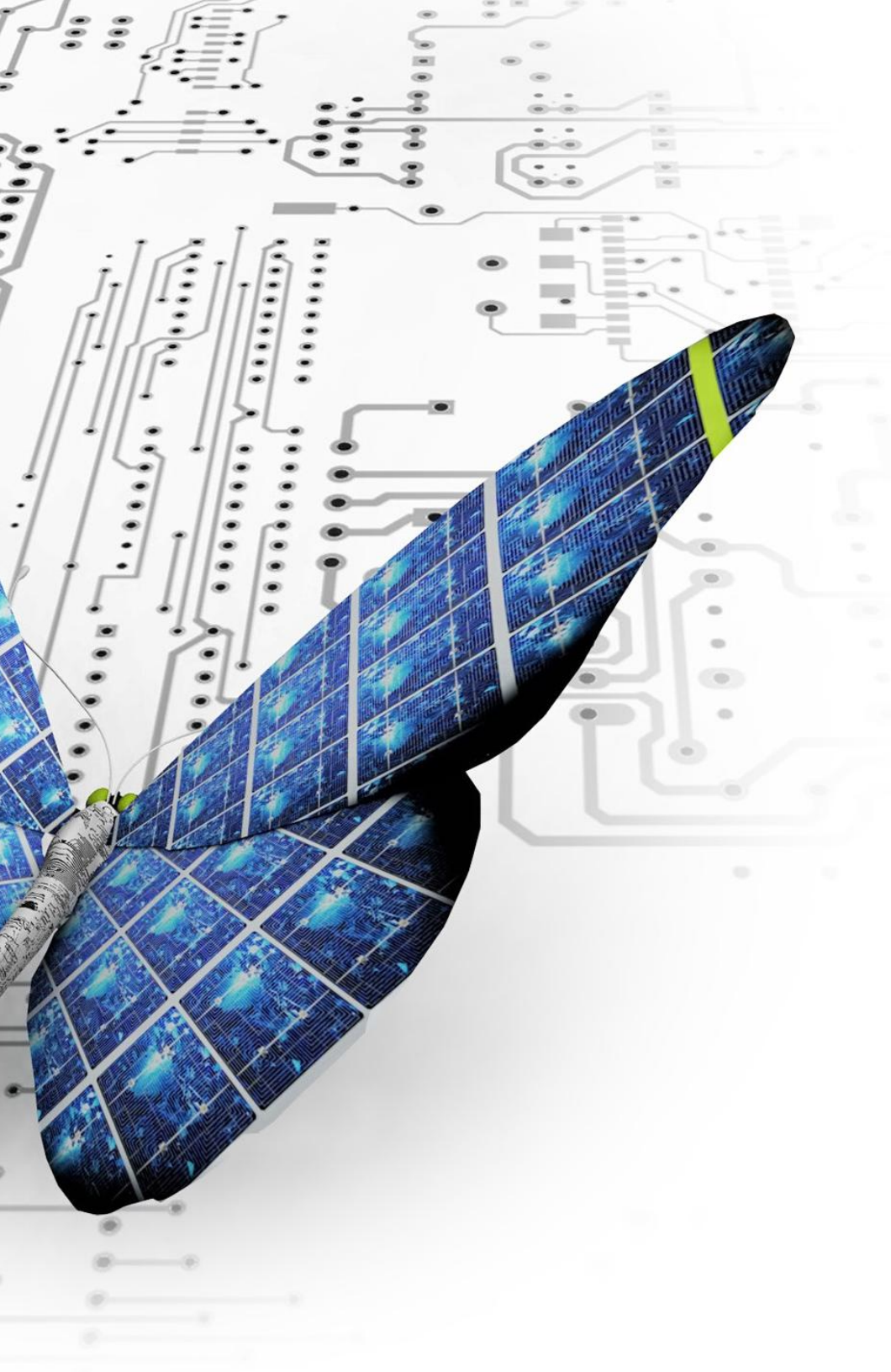
Despite the fact that Kubernetes is excellent at recovery in the event that Pods fail, it is still critical to be able to pinpoint problems in the environment. Therefore, logging events and monitoring the overall health of Kubernetes clusters is key. Simply deploying any open source monitoring and logging tool that leverages the Kubernetes API doesn't completely solve the problem. By its nature, a Kubernetes environment is highly dynamic and consistently yields orders of magnitude more events than does a traditional application environment. Kubernetes administrators therefore grapple with how to make sense of a mountain of data to efficiently identify and remediate issues.

The standard practice for monitoring container-based applications is to log standard output and standard error. However, in cases where a container crashes unexpectedly, you still need some way to access the application logs. This requires a separate backend to store, analyze, and query the logs. These capabilities aren't provided by Kubernetes natively, so a third-party solution is recommended.

Generally, a single tool isn't enough to robustly solve the entire Kubernetes monitoring and logging problem. The most common "stacks" employed to store, search, analyze, and visualize Kubernetes environment data are the ELK (Elasticsearch, Logstash, and Kibana) and EFK (Elasticsearch, Fluentd, and Kibana) stacks. Prometheus is an open-source systems monitoring and alerting toolkit, and is widely used in Kubernetes environments.

Elasticsearch is a distributed, scalable search engine, Logstash/Fluentd are data processing tools, and Kibana is a data visualization and frontend for Elasticsearch. These stacks combine to provide a robust and powerful logging stack.

Properly configuring, sizing, and utilizing logging stacks are an enormous challenge. It's also generally accepted that cluster-level logging, and application logging are separate processes.



CLUSTER SCALING

The dynamic resource management that Kubernetes executes translates to significant efficiency improvements in infrastructure utilization. **Kubernetes is capable of automatically scaling applications, Pods, and clusters.** This degree of flexibility is a huge advantage, but it is also a common point for IT operations to get caught up in figuring out the right scaling approach.

Automatic application scaling

Automated scaling of Kubernetes applications is built directly into Kubernetes, and is handled via horizontal Pod autoscaling. However, it is required that users first ensure that there is enough cluster capacity to support their maximum scaling values.

Automatic worker node scaling

Automatically scaling worker nodes requires a third-party mechanism outside of Kubernetes. For example, this capability would be executed by the infrastructure provider, or by dedicated infrastructure teams on premises. Total infrastructure capacity here is relevant, as you do not want scaling without limits if your cloud has resource limits.

THE CONCLUSION
TO KUBERNETES LIFECYCLE MANAGEMENT

While no two cloud native journeys are exactly alike, they all will involve tackling complexity across a broad spectrum of Kubernetes functionality. The expanding cloud native ecosystem features scores of innovative solutions that can help, especially in the areas outlined above. One important thing to keep in mind - especially when it comes to deploying a lifecycle management solution for Kubernetes - is that proprietary features can degrade the native user experience and make it difficult to port applications across to other Kubernetes environments. Seek out CNCF-certified Kubernetes distribution and solutions. Conformance ensures interoperability!

Fast-track your way to production-ready cloud native infrastructure!

GET STARTED