LEVEREGE

An Introduction to Kubernetes

Table of Contents

Foreword

"Kubernetes is eating the container world."

The hype around Kubernetes continues to grow each day. It remains a top ten open-source project on GitHub. While the development community has embraced the tool as the de facto standard for container orchestration, decision makers outside the container world have yet to learn about the tangible benefits Kubernetes can provide.

So why would anyone care about an infrastructure software tool, especially in the context of IoT deployments?

As it turns out, operational challenges—including deployments, incident management and application management—affect the success of IoT projects just as much as technical challenges regarding devices, networks, etc. Kubernetes excels in abstracting away cloud infrastructure. It solves key pain points, including service discovery, storage orchestration, self-healing, automated upgrades and rollbacks, configuration management, and horizontal scaling. With Kubernetes managing containers and reducing deployment challenges, our team at Leverege was able to focus more on other technical and operation features rather than trying to keep the existing solution up and running.

A lot of the documentation and resources online currently are heavily geared toward developers. While this eBook leans toward the technical side, our objective is to provide enough context and illustrate the benefits to allow your team to consider implementing Kubernetes for all of its benefits. Our eBook is by no means a comprehensive guide on all things Kubernetes but rather a starting point to kick off the discussion on using it to deploy on the cloud or on the edge.

At Leverege, we are committed to amplifying human potential through IoT. We firmly believe that everyone is made better by openly sharing knowledge, so we have created this resource for you. We have seen firsthand that many organizations do not know enough even to begin asking the right questions. We hope to demonstrate the benefits of Kubernetes in the context of IoT.

If you still have questions after reading this eBook, please do not hesitate to reach out. IoT may seem hard, but it does not have to be.

— Team Leverege

1

Introduction to Kubernetes

Introduction to Kubernetes

Kubernetes, or k8s for short, is an open-source container orchestrator. Originally developed by the engineers at Google, Kubernetes solves many problems involved with running a microservice architecture in production. Kubernetes automatically takes care of scaling, self-healing, load-balancing, rolling updates, and other tasks that used to be done manually by DevOps engineers.

Since Kubernetes was open-sourced and managed by Cloud Native Computing Foundation in 2014, the development community has embraced its benefits to orchestrate container-based systems. This book will detail the basics of Kubernetes and some of the lessons that the engineers at Leverege have learned in production.

From Docker to Kubernetes

Docker and Kubernetes are complementary technologies. When Kubernetes orchestrates a new deployment of a container, it will instruct Docker to fire up specified containers. The key advantage of Kubernetes is that it will automatically determine how to schedule and distribute pods (we will go over the terminology shortly), instead of having a DevOps engineer manually asking Docker to do these tasks. Kubernetes is the orchestrator, and Docker is the container runtime engine.

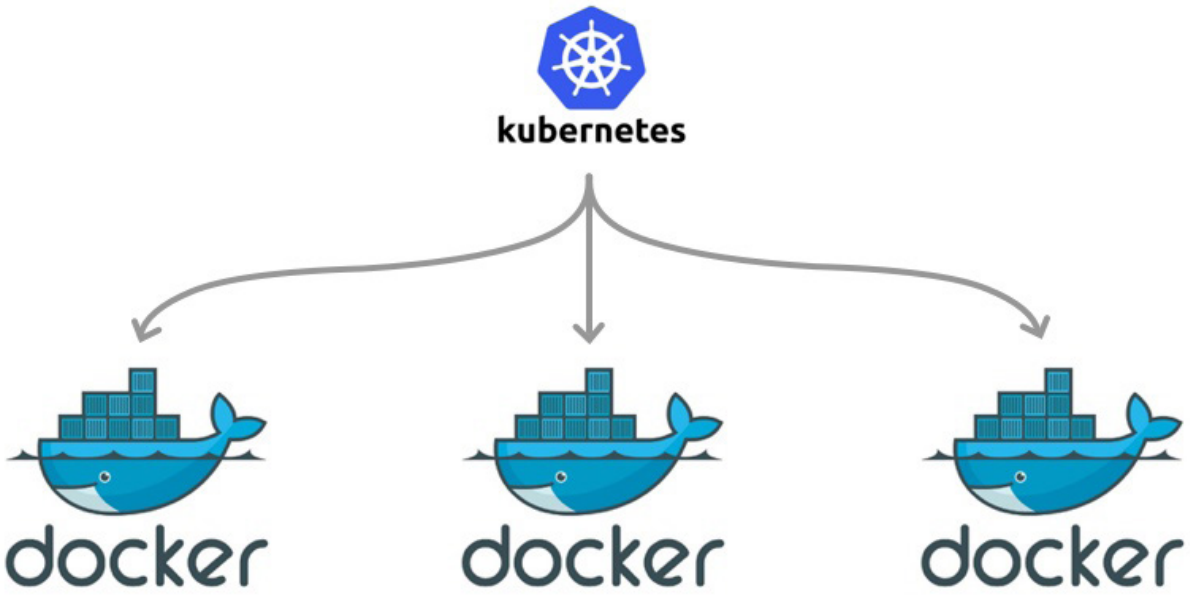


Image Credit: Mesosphere

Kubernetes vs. Docker Swarm vs. Apache Mesos

With Docker now natively supporting Kubernetes, the debate between Kubernetes vs. Docker Swarm has now largely been reduced to personal preference. However, if you are starting from scratch, you may want

to read up on how each of these container orchestration solutions differ in their mission, and also how they play well with each other.

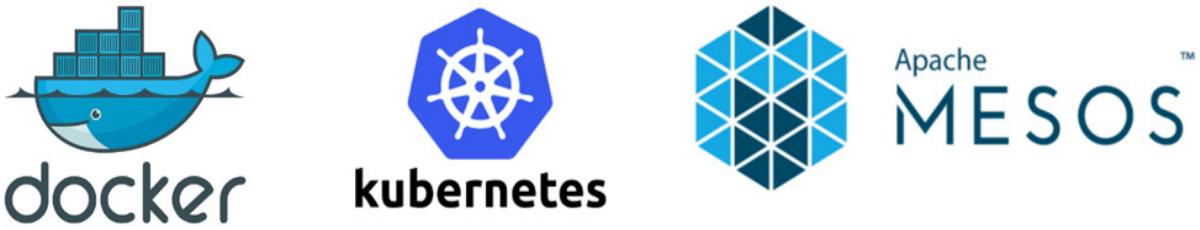


Image Credit: Mesosphere

Docker Swarm

Docker is most well-known for standardizing the container format that solved the issue of managing multiple dependencies and binaries across many application environments. With git-like semantics, Docker images work intuitively and have come to influence Cloud Native Computing Foundation (CNCF) and Open Container Initiative (OCI).

As more and more of the community began to embrace Docker, the need for a container orchestration solution naturally emerged. Before Docker Swarm was introduced, Mesos and Kubernetes were actually recommended by Docker as a container management solution in production.

Since then, Docker has become more than a mere Docker file format provider. It has added Docker hub, registry, cloud, and other services to become a true Platform-as-a-Service. One of the key strengths of Docker Swarm is that if you were already using Docker-compose files, it is really easy to get setup and start using it. However, if you do not want to depend on the entire Docker suite of tools, then Docker Swarm is not as "open" as Kubernetes or Mesos.

Kubernetes

No other company can match Google's extensive experience running billions of containers. So when Google decided to donate Kubernetes—based on their proprietary tool called Borg— to Cloud Native Computing Foundation (CNCF) in 2014, Kubernetes quickly became the de facto standard interface to abstract away the underlying infrastructure.

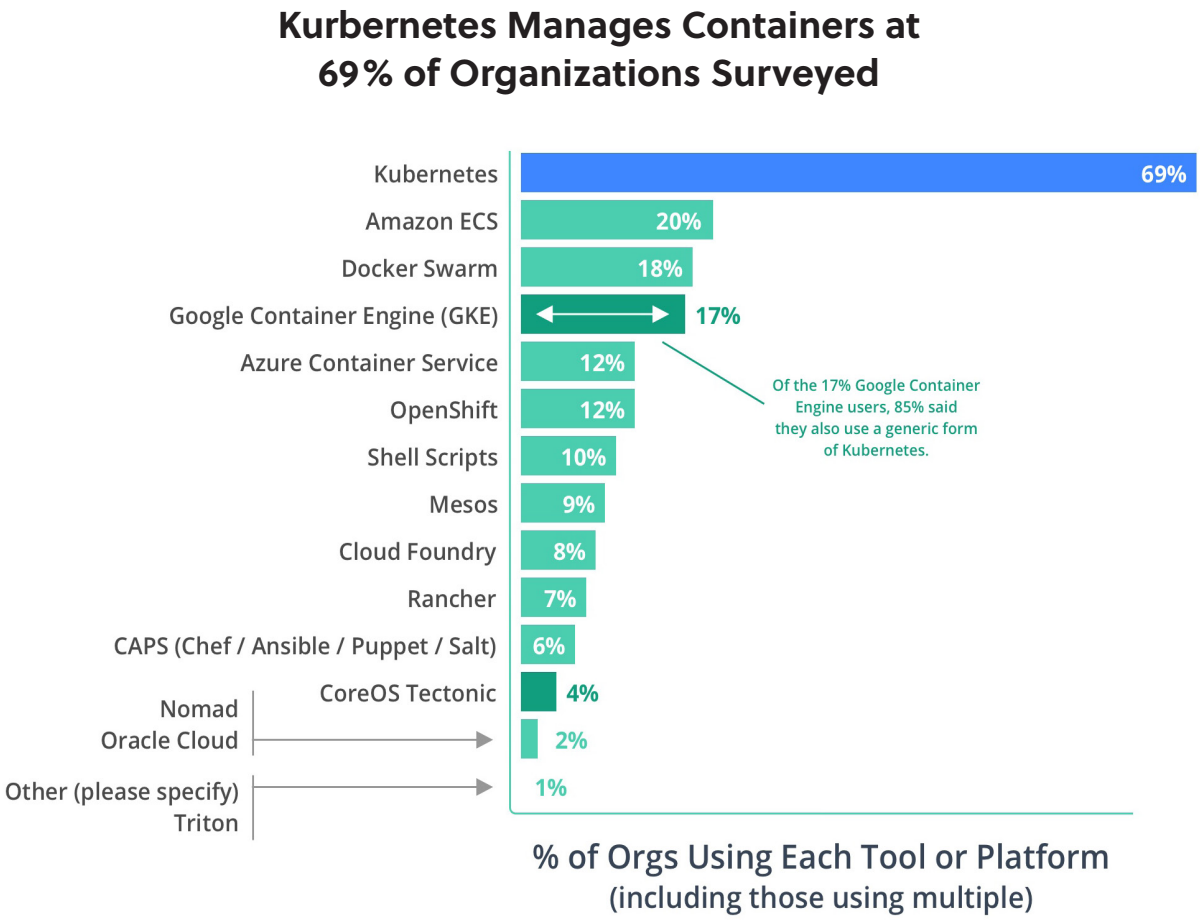


Image Credit: The New Stack

The main strength of Kubernetes is the thriving open-source community led by Google, Red Hat, and CoreOS. It is one of the top open-source projects on GitHub. The rich set of features provided by the core Kubernetes API—as well as tools built on top, such as Helm, Istio, and Spinnaker—allow application developers to focus on code and not as much on infrastructure. Because Kubernetes grew out of Borg to fit containers and Docker into its existing orchestration plans (as opposed to Docker Swarm, which built on top of its Docker container experience to build orchestration), the focus of Kubernetes is less on superior "zero-to-dev" experience than on reliability and portability. To date, Kubernetes is the only solution to enable cluster federation and hybrid cloud models to avoid vendor lock-in.

Mesos

Mesos is the oldest technology out of the three. It is not a true container orchestrator but a cluster manager originally designed to abstract away data center resources and scale diverse workloads that may include stateless microservices (containers), legacy monoliths, batch jobs, or data services.

Mesos shines in managing stateful data workloads like HDFS or Cassandra, where a typical blue/green deployment for microservices may result in data loss. So when people compare Docker Swarm or Kubernetes to Mesos, they are really comparing it to a service within Mesos called Marathon that does orchestration.

Why Is Kubernetes Useful to Me?

Very few companies run at the scale of Google, Netflix, and Spotify, cycling through millions of containers. But even for the rest of us, as we scale the number of containers from a handful to even several tens or hundreds of containers, the need for a container orchestrator becomes clear.

Coupled with the continued rise of cloud computing, Kubernetes serves as a single interface through which to request data center resources. The tangible value of Kubernetes for Leverege manifested in cost-savings for running production loads in spot/preemptible machines while reducing the operational costs of dedicated DevOps resources.

Further Reading:

- Kubernetes vs. Mesos vs. Swarm - An Opinionated Discussion
- Docker vs. Kubernetes vs. Apache Mesos: Why What You Think You Know is Probably Wrong
- Orchestration Platforms in the Ring: Kubernetes vs. Docker Swarm

2

Kubernetes Concepts

Kubernetes Concepts

At a high level, Kubernetes works similarly to many cluster architectures. It consists of one or more masters and multiple nodes that they control. The master nodes orchestrate the applications running on nodes, and they monitor them constantly to ensure that they match the desired state the programmer has declared. In this chapter, we will dive deeper into the key concepts underlying the Kubernetes Architecture. This is by no means a comprehensive overview of every detail; for that, you should read up on Kubernetes's Official Documentation.

Kubernetes Components

As mentioned previously, Kubernetes can largely be divided into Master and Node Components. There are also some add ons such as the Web UI and DNS that are provided as a service by managed Kubernetes offerings (e.g. GKE, AKS, EKS).

Master Components

Master components globally monitor the cluster and respond to cluster events. These can include scheduling, scaling, or restarting an unhealthy pod. Five components make up the Master components: kube-apiserver, etcd, kube-scheduler, kube-controller-manager, and cloud-controller-manager.

- **kube-apiserver:** REST API endpoint to serve as the frontend for the Kubernetes control plane
- **etcd:** Key value store for the cluster data (regarded as the single source of truth)
- **kube-scheduler:** Watches new workloads/pods and assigns them to a node based on several scheduling factors (resource constraints, anti-affinity rules, data locality, etc.)
- **kube-controller-manager:** Central controller that watches the node, replication set, endpoints (services), and service accounts
- **cloud-controller-manager:** Interacts with the underlying cloud provider to manage resources

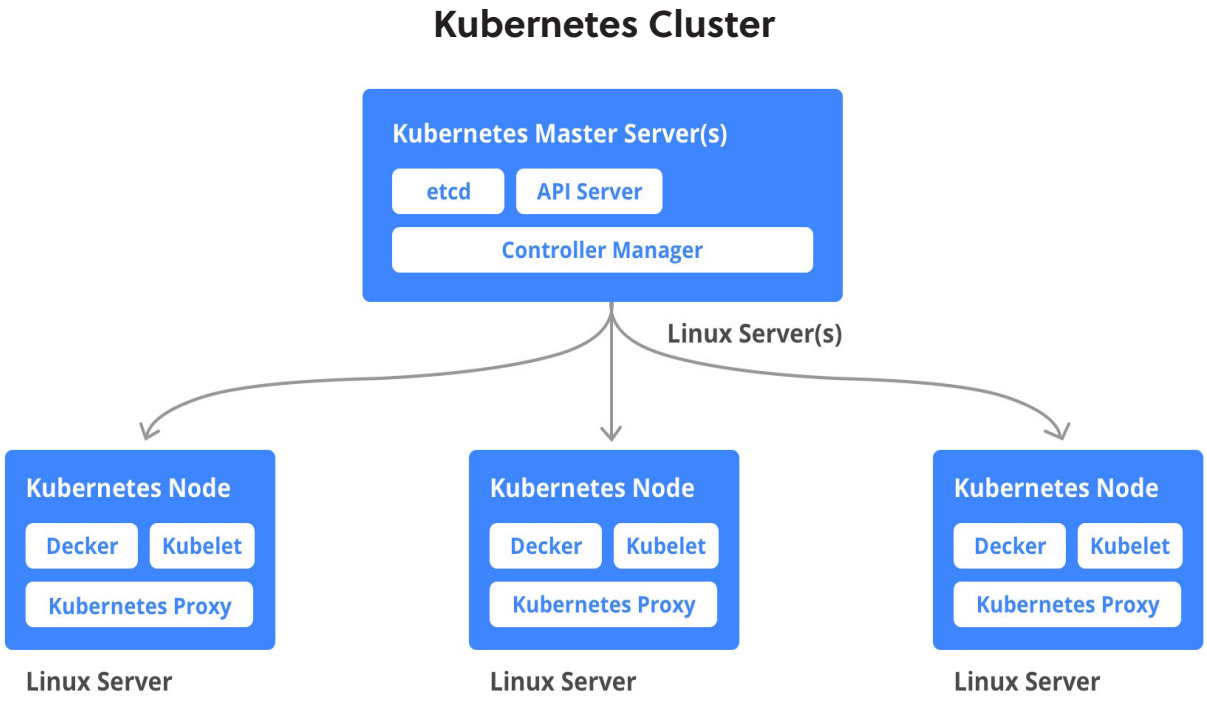


Image Credit: Linux.com

Node Components

Unlike Master components that usually run on a single node (unless High Availability Setup is explicitly stated), Node components run on every node.

- **kubelet:** Agent running on the node to inspect the container health and report to the master as well as listening to new commands from the kube-apiserver
- **kube-proxy:** Maintains the network rules
- **container runtime:** Software for running the containers (e.g. Docker, rkt, runc)

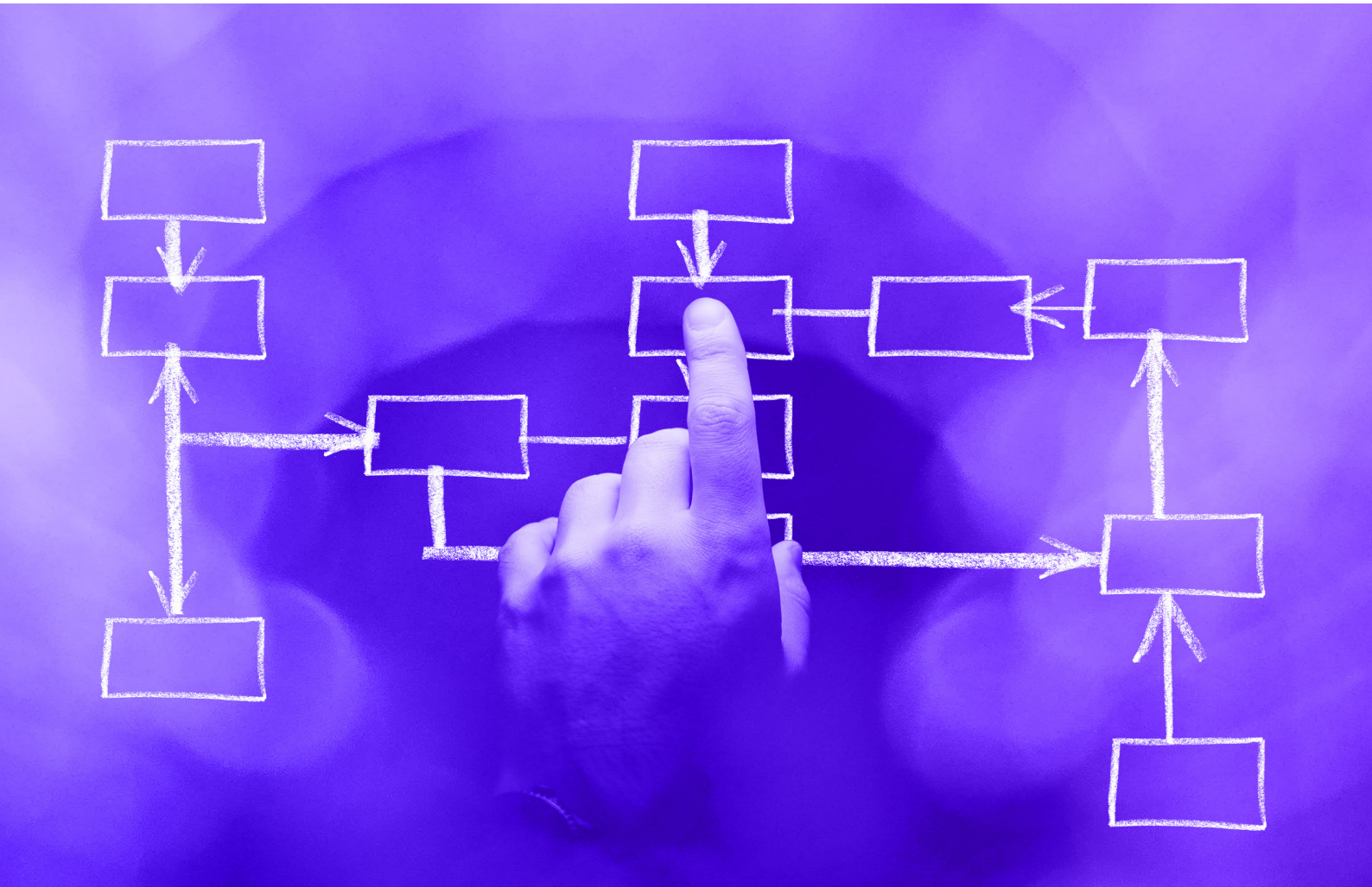
Kubernetes Object Management Model

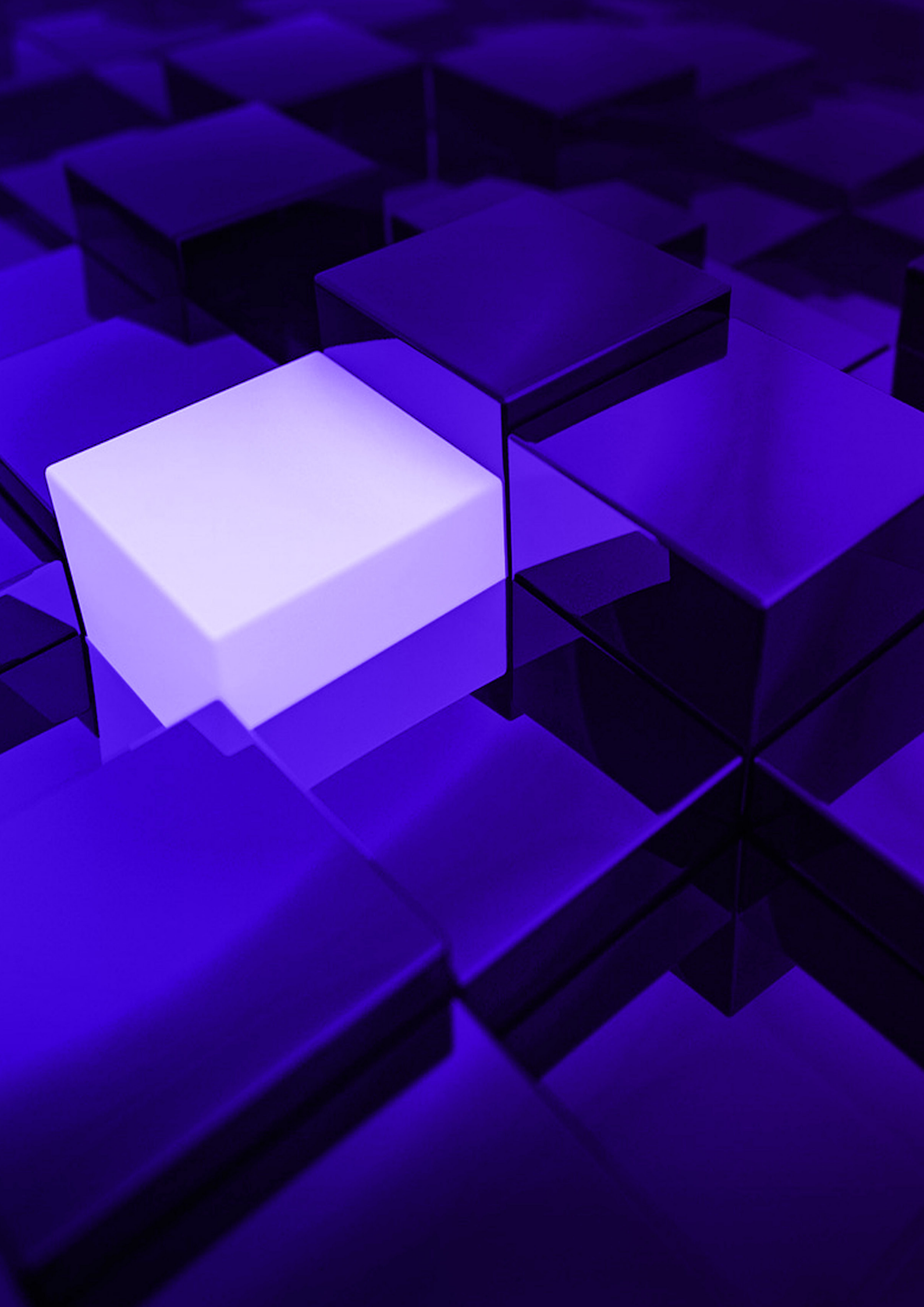
Before jumping into Kubernetes workloads (Pods, Controllers, etc.), it's important to understand Kubernetes's declarative model. Kubernetes actually also has imperative modes, but we will focus on the declarative model and desired states. If you want to learn more, Sebastien Goasguen, the Kubernetes lead at Bitnami, has a great Medium article on the difference between the imperative vs. declarative modes.

Image Credit: The Kubernetes Book

In essence, when we write YAML or JSON files, we describe the desired state of the application: what Docker image should run, what scaling strategy to use, and what ports/services to expose. This information is posted to the kube-api-server, and the master node distributes the work to ensure that the cluster matches the desired state. This configuration is stored in etcd, and the workload is deployed onto the cluster. Finally, Kubernetes will constantly check to see whether the current state of the cluster matches the desired behavior the programmer has defined. So, if a pod dies, Kubernetes will fire up another one to match the desired state.

While this all sounds simple (and that was part of the intent), it is a powerful scheme that makes Kubernetes very useful. You (the programmer) only have to specify the desired state, and Kubernetes will take care of the rest (instead of having you run specific commands to achieve this like in imperative models).





Kubernetes Workloads

Kubernetes workloads are divided into two major components: pods (the basic building block) and controllers (e.g. ReplicaSet, Deployment, StatefulSet, CronJob, etc.).

Pods

A Pod for Kubernetes is what a container is for Docker: the smallest and simplest unit in its object model. It is helpful to conceptualize Pods as a single instance of an application—or a container. In reality, a Pod encapsulates one or more containers as well as storage resources, an IP address, and rules on how the container(s) should run.

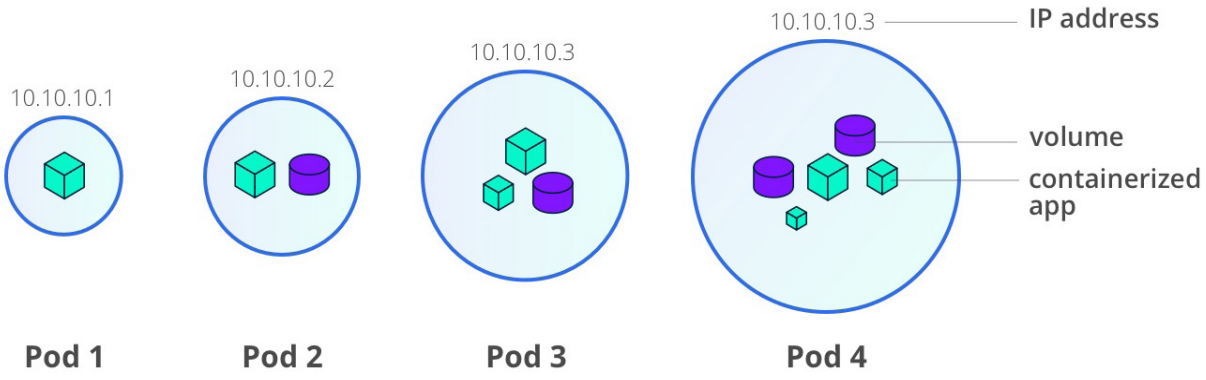


Image Credit: Kubernetes

As Pods are meant to be an atomic unit in Kubernetes, each Pod should really run a single instance of a given application. So, if you need to run multiple copies of a container, each container should run in its own Pod instead of having all of those containers be in a single Pod. However, sometimes a multi-container Pod makes sense if they are closely-related

(a common example is some logging component). One key thing to note is that all the containers in a Pod will share the same environment: memory, volumes, network stack, and most importantly, the IP address.

In a typical deployment, Pods are not directly created by the programmer. Instead, the Controller will schedule a Pod to run on a Node. Some important things to know about Pods:

- A Pod can only exist on a single Node
- A Pod can never be in a partially-deployed state. If a part of the Pod never comes up, it is considered unhealthy and fails.
- A Pod is not "healed" but rather treated as a disposable component. In other words, if a Pod becomes unhealthy, Controller elements will kill the Pod and start up another Pod, replacing rather than healing it.

Controllers

As mentioned earlier, Pods are usually deployed indirectly via Controllers. The one used most frequently is Deployments, but we will quickly cover some other types of Controllers.

ReplicaSet

ReplicaSet, as the name suggests, deploys the specified replicas of the Pod. Unless you require custom updates to the Pods, it is recommended to use Deployments, which are higher level objects that wrap around ReplicaSets.

Deployments

Deployments allow for rolling updates and easy rollbacks on top of ReplicaSets. You can define the desired state in the Deployment model, including scaling, rolling updates in canary or blue/green fashion, and Deployments will take care of it for you.

StatefulSets

StatefulSets are similar to Deployments, but it maintains a "sticky identity" for each of the Pods. It is useful for applications in which a stable network identifier or persistent storage is required. A common example would be ElasticSearch.

CronJobs

As the name suggests, CronJob manages time-based jobs. Going with our ElasticSearch example, one common task would be sending out daily reports or cleaning up old data.

3 Useful Tools

Useful Tools

This chapter lists some useful Kubernetes tools we have used in production. You can follow the official awesome-kubernetes for a comprehensive list, but here is a short list of tools we have vetted and found useful.

Books & Resources

- The Kubernetes Book - Great introduction to Kubernetes concepts
- Kubernetes The Hard Way - Kelsey Hightower's tutorial

Security

- Kubernetes Security Best Practice - Collection of best practices for K8 security
- kubeaudit - Audit tool to scan for common security flaws
- kube-bench - Another audit/scanning tool

Package Manager

- Helm - K8 package manager

Monitoring

- Prometheus-Kubernetes - Great sample Prometheus configs


10mm
250MM LUX-TOOLS

Testing

- Kube Monkey - Implementation of Netflix's Chaos Monkey for K8 clusters
- K8s-testsuite - Load and network test for K8s

CI/CD

- Keel - Lightweight CD that works well with Helm
- Spinnaker - Multi-cloud, battle-tested CD system

Dev Tools

- kube-ps1 - K8 prompts for bash and zsh
- kubectx - kubectl wrapper

Others

- Kubernetes Network Policy Recipes - Great collection of sample network policy recipes
- cert-manager - TLS cert manager for K8s

4 Monitoring

Monitoring

One of the downsides of microservice architecture is increased complexity in monitoring. How does one monitor a cluster of distributed applications that are communicating with each other? First, we need to monitor the health of individual pods and applications. Is the pod scheduled and deployed as intended? Are the applications inside those pods running without errors and without performance degradation? Second, we need to monitor the health of the entire Kubernetes cluster. Is Kubernetes properly handling the resource utilization of each node? What about the health of all of the nodes?

In this chapter, we will examine some of the native tools Kubernetes provides as well as some GCP-specific and open-source tools that we have found useful in production. Please note that this chapter is by no means a comprehensive overview of all available monitoring solutions for Kubernetes users. However, a combination of StackDriver and Prometheus/Grafana have proved to be a robust and reliable tool for our IoT deployments on GKE.

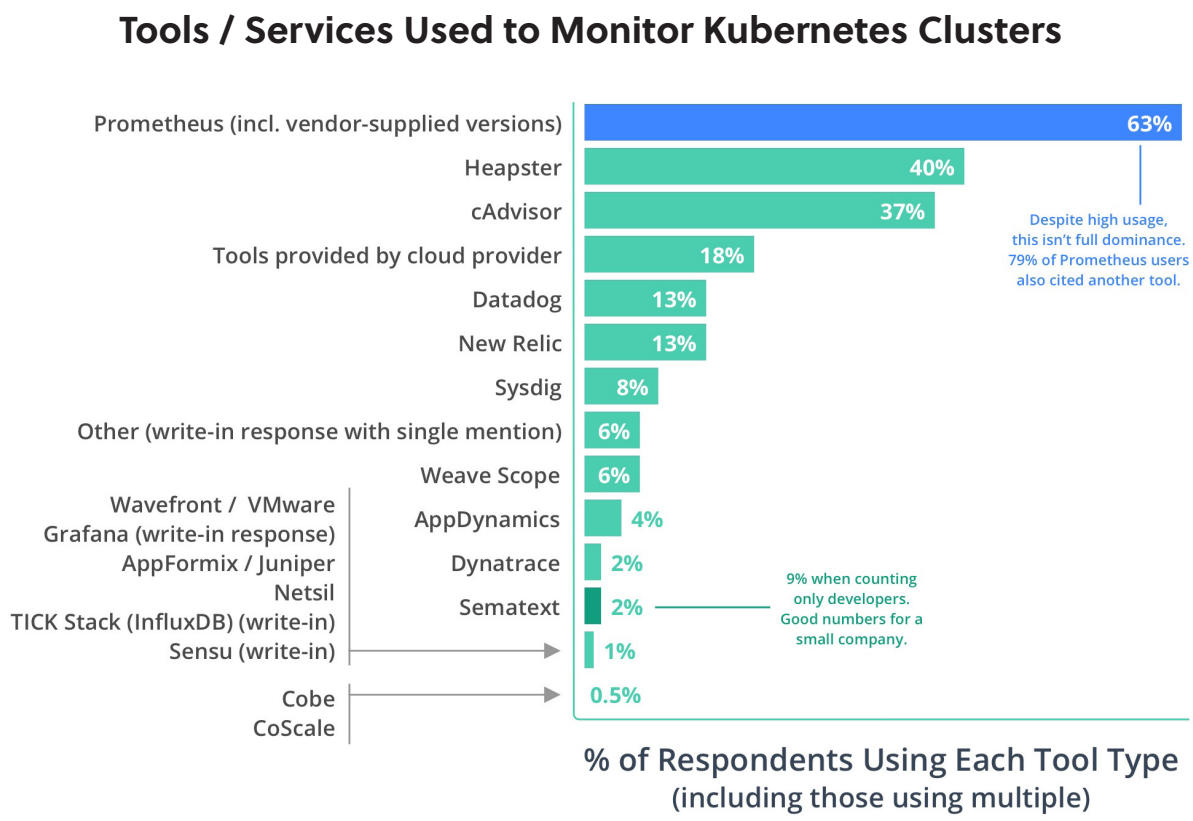


Image Credit: The New Stack

Kubernetes Native Tools

The most rudimentary level of monitoring is administered via native Kubernetes features such as probes, cAdvisor, heapster, and kube-state-metrics. Readiness probe checks the health of a container before a pod is pushed live. Liveness probe periodically monitors the pod by running a defined set of commands to ensure that the pod is running as intended. A failed probe initiates a restart of the pod in an attempt to escape the error state.

cAdvisor and heapster collect container resource usage. cAdvisor oversees all the pods in a node and collects overall machine resource utilization (CPU, memory, filesystem and network usage). Heapster aggregates this data across all the nodes in a cluster to provide an overview of the cluster's health. Lastly, kube-state-metrics provides information via the Kubernetes API. This exports information such as the number of replicas the cluster has scheduled vs. currently available; number of pods running vs. stopped; and number of pod restarts.

While these native tools provide a basic overview of the Kubernetes health, they do not store this data, visualize it in an easily consumable manner. Additionally, they cannot answer application level metrics (e.g. how fast is my elasticsearch query, how many times did my API trigger an error, how many messages are being processed). To address this need, the you will need more powerful tools.

Stackdriver

If you are using the Google Kubernetes Engine, event exporter for Stackdriver Monitoring is enabled by default if cloud logging is enabled. For instructions on deploying to existing clusters, please see the Official Documentation.

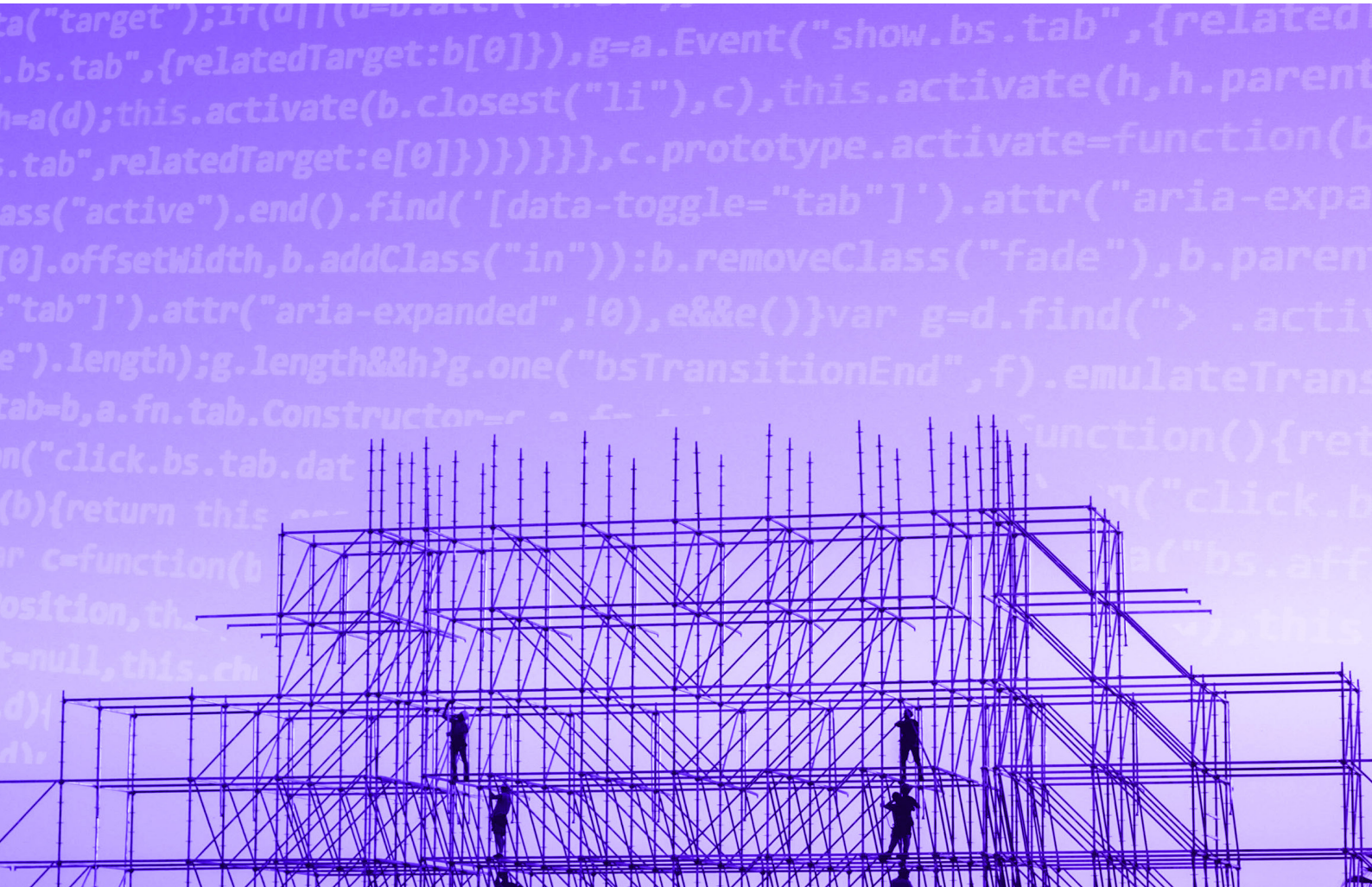
On Stackdriver, you can monitor several things:

- Incidents
- Events
- CPU Usage
- Disk I/O
- Network Traffic
- Pods

Building a dashboard using the above metrics is simple and straightforward, but making adjustments to the default settings to extract specific information is not well supported. If you are looking for a more robust monitoring solution, we recommend a combination of Prometheus and Grafana. It seems like others agree as well.

If you would like to use a combination of Stackdriver and Prometheus, there are tools to export data from each. We will cover exporting Stackdriver metrics to monitor GCP components with Prometheus below. One reason for doing this may be to monitor GCP-related metrics, such as the number of PubSub failed deliveries, BigQuery usage information, and Firebase status.

- Stackdriver to Prometheus Exporter
- Prometheus to Stackdriver Exporter



Prometheus + Grafana

The following guide is heavily inspired by Sergey Nuzhdin's post on LWOLFS BLOG. He does a fantastic job of laying out how to deploy Prometheus and Grafana to Kubernetes using Helm charts. However, at the time of writing this, Nuzhdin does not cover deploying the newer version of Prometheus (2.1) and configuring alerts. To fill this knowledge gap, we created a Prometheus + Grafana Setup Guide that goes through the setup process step by step.

Prometheus

Prometheus is an open-source, time-series monitoring tool developed by the Cloud Native Computing Foundation (CNCF) project—the foundation behind Kubernetes. It is a flexible system that can collect metrics, run complex queries, display graphs, and trigger alerts based on custom rules. Default deployment of Prometheus on Kubernetes scrapes all the aforementioned metrics exposed by probes, cAdvisor, heapster, and kube-state-metrics.

Additional benefits of Prometheus kick in when annotations are added to individual applications to expose custom metrics. As long as an endpoint can be reached with a predefined Prometheus-like format, Prometheus can scrape the metrics and aggregate with other information. The Prometheus server can also be linked to a persistent memory to store all the monitoring metrics. Lastly, Prometheus comes with an Alertmanager that can trigger alerts based on Prometheus-defined rules and notify DevOps engineers via email or Slack.

Monitoring Custom Metrics

The Helm chart for Prometheus will set up the default monitoring metrics using kube-state-metrics (if enabled). But what if you want to monitor your node.js app? You can use a prom-client to format the metrics to be scraped by Prometheus. By default, Prometheus will scrape the '/metrics' endpoint, and the various client libraries will format the outputs to be read by Prometheus.

Monitoring GCP Metrics

Since event exporting is enabled by default with GKE, StackDriver is a great option to monitoring GCP metrics. These include mature products such as PubSub, Compute, and BigQuery, as well as newer products in Cloud IoT, ML, and Firebase. (For a complete list of available metrics, see this documentation.) Exporting these events to Prometheus follows a similar procedure as exporting custom metrics. In fact, GitHub user frodenas has made a Docker image of an exporter available to deploy via Helm/Kubernetes easily.

Grafana

Grafana is an open-source software for time-series analysis and visualization that has native plugins for Prometheus and other popular libraries (Elasticsearch, InfluxDB, Graphite, etc.). While Prometheus provides a rudimentary visualization functionality, it islimited to a simple time-series graph format. Grafana allows for easier visualization of all of the metrics exported by Prometheus to be consumed in various formats: status checks, histograms, pie charts, trends, and complex graphs.

Deploying to Cloud Providers

The State of Managed Kubernetes

Since Amazon Elastic Container Service for Kubernetes (Amazon EKS) went GA in June, all three major cloud providers now support a managed Kubernetes service in production. All three—EKS, Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE)—now also support Kubernetes version 1.10.x, reducing the gap GKE held since Google released what was then called Google Container Engine in 2015.

While the major differences in managed Kubernetes services extend to cloud-level differences from the providers (e.g. Azure is obviously more compatible with Windows-based products), there are several key billing and functionality differences in the three offerings. We will start with EKS, since AWS dominates most cloud computing marketing. We will then work our way backward in terms of date released, namely AKS followed by GKE.

	GKE	AKS	EKS
Offering	GA	GA	GA
K8s Version	1.10	1.9	1.10
Multi AZ	Yes	Partial	Yes
Upgrades	Auto/On-demand	On-demand	Unclear
Auto-scale	Yes	Self-deployed	Self-deployed
RBAC	Yes	Yes	Yes
Network Policy	Yes	Self-deployed	Self-deployed
GPU Support	Yes	Yes	Yes
Persistent Volumes	Block	Block and CIFS	Block
Load Balancer	Yes	Yes	Yes
Mgmt via (vendor) CLI	Complete	Complete	Minimal

Table Data: Hasura

Amazon Elastic Container Service

Although EKS was the latest service to become GA on the market, according to a CNCF Survey in 2017, 63 percent of companies utilizing Kubernetes were already self-hosting Kubernetes on AWS. This high figure means that the barrier to adoption should be low for those existing Kubernetes users. EKS also accelerates the adoption process for those who were not running self-hosted Kubernetes.

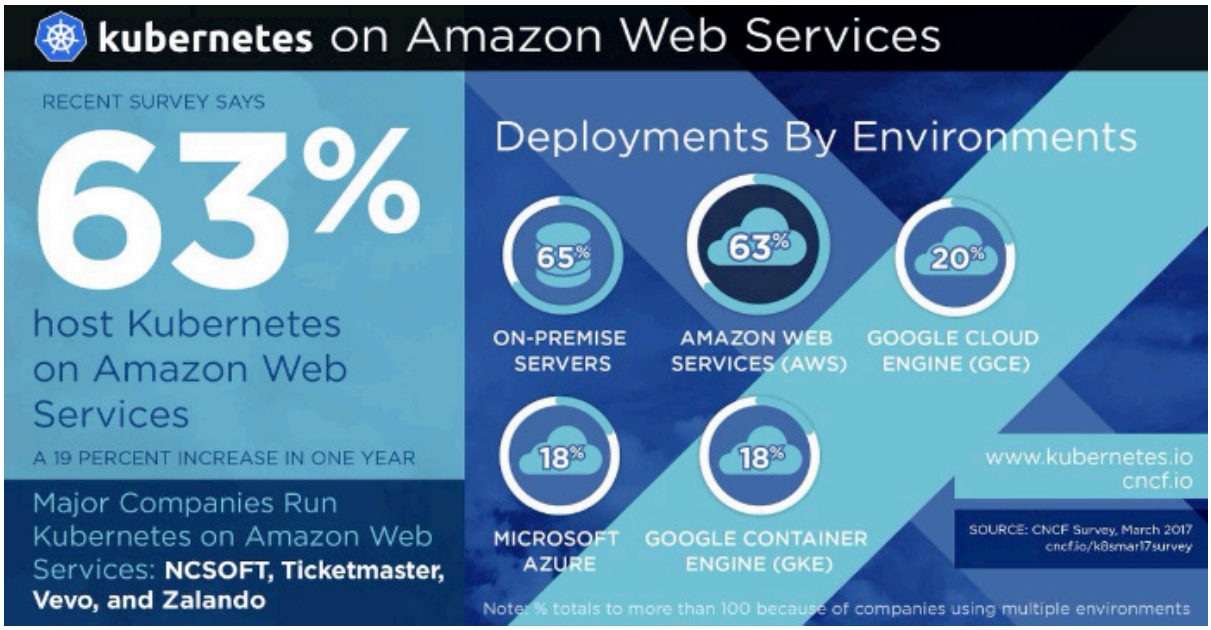


Image Credit: CNCF Survey

The main advantage and potential disadvantage of EKS (and even AWS more generally) is the sheer volume of AWS products. If you or your company is already using other AWS services—e.g. RedShift, Shield, Cloud Watch, or even some IoT products like Greengrass—integrating with EKS is a huge plus. EKS also provides high-availability (HA) modes as well as the most zones available based on the underlying AWS

architecture. However, because AWS had previously focused on its non-Kubernetes version of container orchestration service, Amazon Elastic Container Service (ECS) and Fargate, the community support and involvement is neither as large nor as focused as it is with the other cloud offerings. If you were already using ECS, the incentive to switch to EKS may be small, unless you are exploring hybrid cloud solutions.

EKS is not as intuitive as AKS or GKE for beginners. Instead of using a click-and-deploy model, EKS requires a more custom setup. Namely, IAM role setup must be done outside the EKS initial setup, as well as needing kubectl or CloudFormation setup to add worker nodes to the master node instead of specifying this by default. All of this may be fine for experienced Kubernetes or AWS users, but for a beginner looking to deploy a managed Kubernetes solution, the setup time is significantly higher than AKS or GKE.

Lastly, AWS charges for master node usage. It charges $0.20/hr for the master node plus usage for worker nodes for the cluster. Pricing is always tricky to compare across cloud providers since billing is counted slightly differently (not to mention heavily discounted enterprise deals), but at face-value, EKS is significantly more expensive than AKS or GKE since master node usage is not covered by the service.

Summary:

- **Pros:** Integration with other AWS tools, high number of Availability Zones, easy to migrate or integrate if already using AWS container options
- **Cons:** Expensive, steep learning curve plus setup time, relatively new and lacking certain Kubernetes-specific features

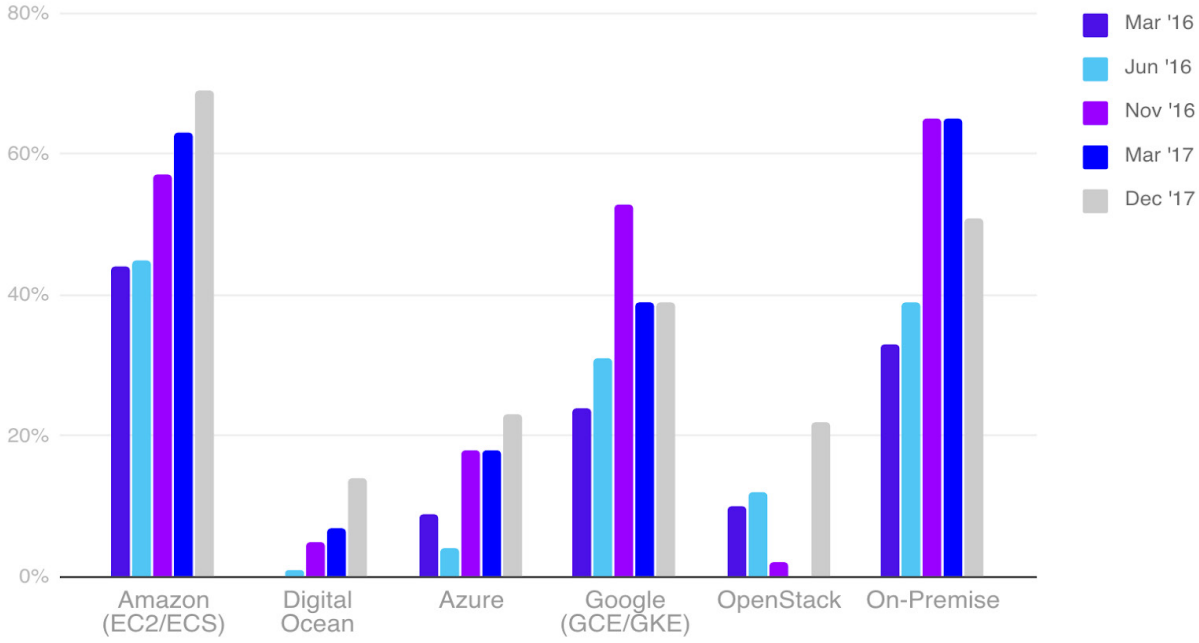
Azure Kubernetes Service (AKS)

Azure released AKS in October 2017, providing managed Kubernetes service for free to Azure users. Unlike AWS, which charges for master node management fee, Azure only charges for Kubernetes VM usage. Similarly to AWS, however, Azure originally had pushed an ECS-equivalent service called Service Fabric. Contrary to a popular misconception that Azure is only compatible with Windows-stack (.NET, C#, etc.), AKS works seamlessly with Azure-hosted Windows and Linux VMs.

Despite an earlier release date than AWS, AKS seems to lag behind AWS and GKE in terms of Kubernetes upgrade, support, and adoption numbers. While EKS and GKE supported Kubernetes 1.10.x relatively quickly since its release, AKS only recently started supporting it. Additionally, unlike EKS and GKE, master nodes are not offered in HA mode. Considering pre-existing usage of Kubernetes on AWS and GKE's early lead in Kubernetes development, community involvement in AKS seems to be low so far.

That is not to say that AKS features are lagging in all aspects. AKS is currently the only provider for an incubator feature called Service Catalog, which allows for applications inside Kubernetes to use externally managed services (e.g. Azure databases). Microsoft has also been active in maintaining a popular Kubernetes package manager, Helm.

Breakdown of Environments Where Companies / Organizations are Deploying Containers



Data Source: IT Pro Today

Summary:

- **Pros:** Works well with Windows VMs, only charged for VM usage (not master node)
- **Cons:** Lags newest features/releases, smaller Kubernetes community on Azure

Google Kubernetes Engine (GKE)

With the exception of perhaps Netflix and Spotify, no other company has as much experience running containers in production as Google. As the original creator of Kubernetes and a huge open-source contributor along with Redhat, GKE brings the best user experience and added features. Unlike EKS and AKS, on GKE, you can easily customize a GKE deployment in various node pools and in HA by clicking a few buttons. Like AKS, GKE only charges for VM usage; generally, with sustained usage fees, GKE comes out to be the cheapest managed Kubernetes service of the three.

Aside from cost, the main advantage of GKE will always be fast access to the newest features and tools. GKE dashboard—along with Stackdriver logging and monitoring agents already embedded on its VMs—allows for easy monitoring of cluster health and usage. GKE also auto-scales your nodes for you, which is a really nice feature that requires custom setup on EKS and AKS—this feature is great for quick prototyping usage or load testing). Overall, GKE abstracts away a lot of the infrastructure setup, simplifying the onboarding experience for new users.

Ultimately the downsides of GKE come not from the technology, but more so from the general lack of other tools and community support on GCP itself. Whereas AWS boasts IAM and Azure touts Active Directory, GKE has no real comparative product. In terms of enterprise support, GCP still lags behind AWS, whose obsession with customers led them to the top, and Azure, whose roots in the Windows ecosystem enabled it to leverage deep ties to enterprise IT infrastructure.

Summary:

- **Pros:** Lowest cost, experienced team-leading Kubernetes development, access to the newest feature
- **Cons:** GCP usage is lowest amongst the big three cloud providers

6

Running GKE in Production

Running GKE in Production

After comparing the managed offerings from AWS, Azure, and GCP, Leverege chose GKE to run its large-scale IoT solutions in production. Every organization's needs are different, but given the unique constraints that the current IoT landscape poses, here's why Leverege chose GKE and what we've learned.

Cost

First and foremost, GKE provides the best price for any managed Kubernetes service. Unlike EKS, which charges for the management of master nodes ($0.20/hr), GKE and AKS only charge for the VMs running the K8 nodes. This means that master node usage, cluster management (auto-scaling, network policy, auto-upgrades), and other add-on services are provided free of charge. Due to Google's automatic sustained use discount and the option to use pre-emptibles, we found that GKE ends up being slightly cheaper than AKS.

(Disclaimer: Pricing comparisons across the three vendors are tricky since enterprise discounts, exact configuration of machines, and storage/GPU needs vary significantly. But in general for a standard setup, Leverege found GKE to be the cheapest.)

Lower cost is extremely important for price-sensitive IoT projects, especially for those using a LPWA (low-power, wide area) network. Let's take a tank monitoring solution for example. A typical deployment would have a small sensor reporting to the network a few times a day with a very small payload to conserve power. From our experience, customers are only willing to pay

around $1 monthly per sensor to justify their ROI on a simple IoT system. Once the hardware and network costs are factored in, the budget left for cloud infrastructure becomes increasingly constrained. This is even before factoring in operational cost unrelated to pure technical support.

Superior Onboarding & Management Experience

When you are paying for a managed service, you expect a seamless onboarding experience and minimal intervention on your part to continually manage the infrastructure. With GKE, you get the best experience regarding Kubernetes management.

To begin, you can easily spin up a cluster with a few clicks. After enabling the Kubernetes Engine API, you can specify the VM type and the number of nodes and click create. GKE takes care of regional deployment, auto-scaling, auto-upgrades of both master and child nodes, as well as network policy configuration via Calico. While these are all things you can self-manage and deploy on AKS and EKS, it is a free service provided and managed by Google so you can focus on application development on top of Kubernetes.

The best part of GKE, in our experience, is the cluster autoscaler feature. GKE will automatically add or delete nodes when Kubernetes can no longer schedules Pods on the existing nodes. Cluster autoscaler is tied to a node pool, so you can autoscale different node pools when needed. This feature really shines when you run a mixed node pool of pre-emptibles and nodes reserved for intensive data workloads. Perhaps you elect to run a node pool of pre-emptibles for non-critical workflow that can scale on demand, while keeping some data-intensive or stateful

sets on another node pool. GKE will be Kubernetes-aware and scale automatically for you, which is not provided by default on EKS or AKS.

Another feature we like about GKE is its close integration with Stackdriver, which has come a long way in recent years. With Stackdriving agents already installed onto GKE nodes, you can easily use preset GKE monitoring charts and logging before electing to set up more custom systems (e.g. Prometheus/Grafana, ELK).

Leading Contributor to K8 & Related Work

Before Google open-sourced Kubernetes, it was running Borg and Omega internally for years. Back in 2014, Google shared that they were already firing up over two billion containers per week. No other player can match Google's 10+ years of container management at that scale. This is most evident in the number of new features that get rolled into GKE before EKS or AKS. Oftentimes the newest versions of Kubernetes are available on GKE only, not to mention the beta features including TPU support.

Google's work also shines in K8-related technologies. Google has been an important contributor to Spinnaker, a battle-tested Continuous Deployment (CD) provider, as well as Istio, a cloud-native service mesh developed with IBM and Lyft. Google is also active on toolkits and extensions for machine learning on K8 (kubeflow) as well as supporting serverless workloads. As the announcement from Google NEXT 2018 showed, Google is committed to growing the Kubernetes community and is a great platform to start that journey.


Google
Ireland
Google Search
I'm Feeling Lucky
Google.ie offered in: Gaeilge

Lessons Learned

After deploying some of the largest IoT systems in North America for over a year, Leverege learned some lessons with Kubernetes the hard way. Due to a huge community that Kubernetes has garnered, you can now easily find lots of tips online. However, we recommend that you start simple. Before messing around with complex namespace configurations, Spinnaker/Istio, and pod policies, start with the default namespace and learn the basics before you secure your cluster for production.

Use Preemptibles and Node Pools

Google provides preemptible VMs that provide no availability guarantees and last for 24 hours maximum at a heavy discount (~5x cheaper). Although preemptibles were initially marketed for running short term batch jobs, with multiple node pools running on GKE, you can run production loads at scale on a mix of preemptibles and reserved instances. Using node pools, we can leverage the power of Kubernetes to schedule pods based on need.

As long as stateless applications can gracefully exit when the preemptible VMs send a SIGTERM signal, we can allow Kubernetes to schedule pods to another available preemptible VM. In the case when another preemptible VM is unavailable (either during an auto-scale event or after the termination signal was given to a prior node), we can schedule these workloads to a reserved instance. For critical or stateful sets, we can define node affinity values to always schedule them on reserved instances. Just by setting node affinity configs, you can start saving cost on your clusters.

Always Know Your Context

Whether you are using Helm or explicitly using kubectl commands, it is easy to forget your context and accidentally push the wrong environment to the wrong cluster. Safeguard yourself from these accidents by always specifying the context/cluster to which you want to push changes. You can also use command line helpers like this one when you start to have more than several clusters.

Manage Complexities

One of the downsides to Kubernetes, or perhaps a microservices architecture more generally, is how quickly complexities can grow. You will need a good plan to manage complexities introduced by more layers or monitoring of your cluster, nodes, pods, and applications will suffer. Start out with consistent logging practices. Either pass context between services or use a tracing system for better observability. Needless to say, this will be extremely helpful when a cluster enters an unknown state and you can track down the message flow easier.

There are various tools to help tame Kubernetes. At Leverege, we use Helm as a templating tool to package Kubernetes manifests as charts. Until you have a CD system set up, Helm can help with rolling updates and easy rollbacks. If you don't have a monitoring system set up, become familiar with the tools Google provides by default. Familiarize yourself with cloud shell and kubectl commands to quickly diagnose and modify your manifests.

Static IPs for Ingress

When creating your Deployment or Ingress templates on Kubernetes, it is easy to forget to reserve static IPs for your public endpoints. The best practice is not to use the type LoadBalancer and use a small number of Ingress endpoints, but if IPs are not reserved or specified, Kubernetes might assign them to a new endpoint that you might not expect.

Set Update Windows

One of the nice features of GKE is auto-upgrades. If you are running a regional deployment, you can upgrade your master nodes without downtime. However, always set your update windows so you can monitor changes in production. There might be hidden, unintended consequences to your key management, logging, or authentication schemes that might break due to an upgrade.

Conclusion: Kubernetes Works for IoT Deployments

As the hype continued to mount for Kubernetes within the development community, there has been some valid criticisms against Kubernetes for its huge learning curve and overhead required to maintain the beast. Very few companies run into the Google-scale problems for which Kubernetes was designed to solve. Nonetheless, we at Leverege have reaped tangible benefits from Kubernetes in terms of autoscaling, self-healing capabilities, immutability, cross-platform compatibility, and easier deployment once the infrastructure has been established.

Given the unique challenges posed by the nascent IoT world, Kubernetes has largely been worth the Leverege team's investment in reducing the resources devoted to DevOps (deployment, monitoring, incident management). As more and more tools become standardized within the Kubernetes ecosystem to improve security, visibility, and usability, Leverege remains confident in the value of Kubernetes for production grade IoT deployments.

iot for all

LEVEREGE

Thanks for reading our eBook!

Like what you saw? Want to find out more?
Connect with our team of experts below.

Connect with Experts