Fairwinds

WHITE PAPER

Kubernetes Best Practices

2021

TABLE OF CONTENTS

Introduction....................3
Security Best Practices....................4
Kubernetes Best Practices for Cost Optimization....................12
Reliability Best Practices....................17
Policy Enforcement Best Practices....................24
Monitoring & Alert Best Practices....................28
Conclusion....................33

INTRODUCTION

In the ever-expanding cloud native ecosystem, often organizations embark on their Kubernetes journey unsure as to what path to follow. Covering all your bases and avoiding common pitfalls and mistakes are worthy goals. No one wants to make the wrong decision and pay for it in the future.

THERE IS NO ONE “RIGHT” PATH TO KUBERNETES SUCCESS; INSTEAD, SEVERAL GOOD PATHS EXIST. WHICH PATH SHOULD YOU FOLLOW? THE ONE THAT BEST ADDRESSES THE NEEDS AND PRIORITIES OF YOUR BUSINESS.

Are you in the finance or healthcare sector where security is non-negotiable?

Do you have a team of busy data scientists or machine learning workloads that require your business to operate with the utmost resource efficiency?

Can your applications and services tolerate downtime, or is 99.99% (or higher) reliability paramount?

Answering these questions (and dozens more) will help you decide how to implement Kubernetes, create process and clarify tasks and priorities. After you have a handle on the bigger picture in your Kubernetes journey, you’ll be better prepared to dig into the inventory of choices and best practices available to you.

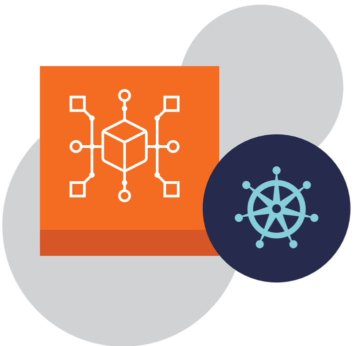
Fairwinds provides software and managed services to help companies ship cloud native applications faster, more cost effectively and with less risk. With years of experience and thousands of customer production deployments, Fairwinds enables companies to confidently run containerized applications on production-grade Kubernetes infrastructure at scale.

Since inception, we’ve helped our clients adopt cloud native infrastructure in a secure, efficient, and reliable fashion. We’ve seen hundreds of different use cases and transformations and guided our client’s on their journey to production.

This paper provides hard won Kubernetes expertise. We dive into the core areas of Kubernetes: security, efficiency, and reliability. Our goal is to provide you with Kubernetes best practices for adoption and implementation so you can realize long-term value across your entire organization.

SECURITY BEST PRACTICES

Avoid Secure by Default Missteps



Kubernetes is becoming a **mainstream solution for managing how stateless microservices run in a cluster** because the technology enables teams to strike a balance between velocity and resilience.

ORGANIZATIONS CONTINUE TO BE INCREDIBLY VULNERABLE TO THREE COMMON THREATS:

1. Denial-of-service (DoS) attacks can take down sites or result in exorbitant computing costs
2. External threats can exploit application code to gain access to internal resources
3. Internal threats can give individual employees access to resources outside the scope of their day-to-day work

As organizations transition to cloud native technologies like containers and Kubernetes, the core business challenge remains the same: figuring out how to accelerate development velocity while maintaining security. Even in the world of Kubernetes and containers, these two business objectives are still in tension.

Kubernetes is becoming a mainstream solution for managing how stateless microservices run in a cluster because the technology enables teams to strike a balance between velocity and resilience. It abstracts away just enough of the infrastructure layer to enable developers to deploy freely without sacrificing important governance and risk controls.

But all too often, those governance and risk controls go underutilized. Since everything is working, it's easy to think that there aren't any problems. It's not until you get hit with a DoS attack or a security breach that you realize a Kubernetes deployment was misconfigured or that access control wasn't properly scoped. Running Kubernetes securely is quite complicated, which can cause headaches for development, security, and operations folks.

KUBERNETES SECURITY CHALLENGES AND BENEFITS

Development teams new to Kubernetes may neglect some critical pieces of deployment configuration. For example, deployments may seem to work just fine without readiness and liveness probes in place or without resource requests and limits, but neglecting these pieces will almost certainly cause headaches down the line. And from a security perspective, it's not always obvious when a Kubernetes deployment is over-permissioned—often the easiest way to get something working is to give it root access.

Security will always make life a bit harder before it makes it easier. Organizations tend to do things in an insecure way by default, because they don't know what they don't know, and Kubernetes is full of these unknown unknowns. It's easy to think your job is done because the site is up and working. But if you haven't tightened up the security posture in a way that adheres to best practices, it's only a matter of time before you start learning lessons the hard way.

Fortunately, Kubernetes comes with some great built-in security tooling, as well as a robust ecosystem of open source and commercial solutions for hardening your clusters. A well-thought-out security strategy can enable development teams to move fast while maintaining a strong security profile. Getting this strategy right is why DevSecOps is so important for cloud native application development.

Furthermore, Kubernetes helps security teams formulate a coherent strategy by putting many pieces of computing infrastructure in one place. This makes it much easier for security teams to conceptualize and address potential attack vectors. The pre-Kubernetes attack surface—the number of different ways to break into your infrastructure—is substantially larger than the Kubernetes attack surface. With Kubernetes, everything is under one hood.

Optimizing Kubernetes security, however, is no easy feat, as there's not one way to handle security in Kubernetes. While it's best to keep people out of the cluster altogether, that goal is hard to achieve since your engineers need to be able to interact with the cluster itself, and your customers need to be able to interact with the applications the cluster is running.

Kubernetes can't secure your application code. It won't prevent your developers from introducing bugs that result in code injection or a leaked secret. But Kubernetes can limit the blast radius of an attack: proper security controls will restrict how far someone can get once they're inside your cluster. For instance, say an outside attacker has found a vulnerability in your application and gained shell access to its container. If you have a tight security policy, they'll be stuck—unable to access other containers, applications, or the cluster at large. But if the container is running as root, has access to the host's filesystem, or has some other security flaw, the attack will quickly spread throughout the cluster. In essence, a well-configured Kubernetes deployment provides an extra layer of security.

BELOW, WE HIGHLIGHT THE FOLLOWING KEY KUBERNETES BEST PRACTICES RELATED TO SECURITY:

- DoS protection
- Updates and patches
- Role-based access control (RBAC)
- Network policy
- Workload identity
- Secrets

DoS Protection

With Kubernetes you can make sure your applications respond well to bursts in traffic, both legitimate and nefarious. The easiest way to take down a site is to overload it with traffic until it goes down—an attack known as denial-of-service (DoS). Of course, if you see a giant burst of traffic coming from one user, you could just shut off their access. But with a distributed-denial-of-service (DDoS) attack, an attacker who has access to many different machines (which they've probably broken into) can bombard a website with seemingly legitimate traffic. Sometimes these “attacks” aren't even nefarious—it might just be one of your customers trying to use your API with a buggy script.

Kubernetes allows applications to scale up and down in response to increases in traffic. That's a huge benefit as increases in traffic (legitimate or nefarious) won't result in end-users experiencing any degradation of performance. But, if you are attacked, your application will consume more resources in your cluster and you'll get the bill.

While services like Cloudflare and Cloudfront serve as a good first line of defense against DoS attacks, a well designed Kubernetes ingress policy can add a second layer of protection. To help mitigate a DDoS threat, you can configure an ingress policy that sets limits on how much traffic a particular user can consume before they get shut off. You can set limits on the number of concurrent connections; the number of requests per second, minute, or hour; the size of request bodies; and even tune these limits for particular hostnames or paths.

tl:dr: set limits!

Updates and Patches

Kubernetes comes out with a few releases a year, each of which fixes bugs and security holes. As painful as upgrading can be, keeping your Kubernetes version up to date is essential. Old versions quickly become stale, and new security holes are being announced all the time.

As painful as upgrading can be, **keeping your Kubernetes version up to date is essential.** Old versions quickly become stale, and new security holes are being announced all the time.

On top of that, it's common to have several add-ons installed in your cluster to enhance the functionality Kubernetes provides out of the box. For instance, you might use cert-manager to help keep your site's external certificates up to date, Istio to handle mutual TLS encryption inside your cluster, or metrics-server and Prometheus to gather metrics about how applications are running. With each of these add-ons, your attack surface and your risks increase. Staying up to date on bug fixes and new releases is important.

Each time a new release comes out, you'll need to test those updates to make sure they don't break anything. Where possible, test on internal and staging clusters and roll updates out slowly, monitoring possible problems and making course corrections along the way.

Finally, be sure to keep the underlying Docker image up to date for each of your applications. The base image you're using can go stale quickly, and new Common Vulnerabilities and Exposures (CVEs) are always being announced. To fight back, you can use container scanning tools like Trivy to check every image for vulnerabilities. But making sure the base operating system and any installed libraries are up-to-date is the safest policy.

tl:dr base operating systems and any install libraries need to be up-to-date and tested thoroughly.

RBAC

The easiest way to deploy a new application or provision a new user is to give away admin permissions. A person or application with admin permissions has free range to do whatever they want—create resources in the cluster, view application secrets, or delete an entire Kubernetes deployment. The problem is that if an attacker gains access to that account, they too can do anything they want. They could spin up new workloads that mine bitcoin, access your database credentials, or delete everything in the cluster.

If you've got an application that doesn't need extensive control over the cluster, giving it admin-level access is quite dangerous. If all it needs to do is view logs, you can pare down its access so that an attacker can't do anything more than that—no mining bitcoin, viewing secrets, or deleting resources.

To manage access, Kubernetes provides role-based access control (RBAC). RBAC is used to grant fine-grained permissions to access different resources in the cluster. Setting up thoughtful Kubernetes RBAC rules according to the principle of least privilege is important for reducing the potential for splash damage when an account is compromised.

It's a delicate balance, as you might end up withholding necessary permissions. But it's worth that minor inconvenience to avoid the major headaches that come from a security breach.

While RBAC configuration can be confusing and verbose, tools like rbac-manager can help simplify the syntax. This helps prevent mistakes and provides a clearer sense for who has access to what.

tl:dr set up RBAC according to the principle of least privilege.

Network Policy

Network policy is similar to RBAC, but instead of deciding who has access to which resources in your cluster, network policy focuses on who can talk to who inside your cluster. In a large enterprise, dozens of applications may run inside the same Kubernetes cluster, and by default every application has network access to everything else running inside the cluster. Of course, some network access is usually necessary. But while a given workload might need to talk to a database and a handful of microservices, that workload probably won't need access to every other application inside the cluster.

It's up to you to write a network policy that cuts off communications to unnecessary parts of the cluster. Without a strict network policy, an attacker will be able to probe the network and spread throughout the cluster. With proper network policies in place, however, an attacker who gains access to a particular workload will be restricted to that one workload and its dependencies.

Network policy can also be used to manage cluster ingress and egress—where incoming traffic can come from and where outgoing traffic can go. You can make sure internal-only applications only accept traffic from IP addresses inside your firewall and make sure all partner IP addresses are whitelisted for partner-driven applications. For outgoing traffic, you may also want to whitelist allowed domains. This way, if a hacker gains access to the cluster and tries to push data out to an external URL, they'll be stopped by your network policy. With strict ingress and egress rules, you can limit the potential attack surface of your applications.

Network policy is easy to neglect, especially as you're building out a Kubernetes cluster for the first time. But it's a good way to harden your cluster from a security standpoint and limit the extent of damage after attackers find a security hole. As with RBAC, there's a tradeoff between over-permissioning to make sure everything works properly versus limiting permissions and making sure any problems are contained. Again, you're sacrificing short-term convenience to avoid the fallout from a major security breach.

tl:dr write a network policy that cuts off communications to unnecessary parts of the cluster. Manage cluster ingress and egress based on IP addresses.

Workload Identity

Workload identity is a way to tie RBAC, the cluster's authentication mechanism, to the cloud provider's authentication mechanism, like Identity and Access Management (IAM) on Google Cloud or AWS. In this way, you can use Kubernetes' built-in authentication mechanisms to manage access to resources that live outside the cluster. For example, databases typically live outside of the Kubernetes cluster in a managed service like AWS's Relational Database Service (RDS). Workload identity allows a workload in your EKS cluster to connect to your RDS instance without you having to provision and manage the credentials yourself.

Without workload identity, you'd have two options, both of which have security concerns. First, you could use IAM to grant the necessary permissions to entire nodes, but this effectively grants those permissions to every workload on the node, not just the workload that needs them. Alternatively, you could generate a long-lived access key for your database, turn that key into a Kubernetes secret, and attach that secret to the workload. But each step in this process opens up the potential for leakage, and because the key is long-lived, anyone with access to that key would be able to access your database in perpetuity.

With workload identity, Google handles all the permissioning under the hood using short-lived credentials, so you don't need to manage and possibly expose your access keys.

With workload identity, Google handles all the permissioning under the hood using short-lived credentials, so you don't need to manage and possibly expose your access keys.

tl:dr: Employ workload identity to tie RBAC to the cloud provider's authentication mechanism.

One caution: workload identity works only within a particular cloud provider. For instance, Google Kubernetes Engine can use workload identity to authenticate databases on Google Cloud but not on AWS; Amazon EKS, in turn, can use workload identity to authenticate AWS databases but not Google Cloud databases.

Secrets

Kubernetes empowers Infrastructure as Code (IaC) workflows more than any other platform. By encoding all of your infrastructure choices in YAML, Terraform, and other configuration formats, you ensure your infrastructure is 100% reproducible. Even if your cluster disappeared overnight, you'd be able to recreate it in a matter of hours or minutes so long as you're utilizing IaC.

But there's one catch: your applications need access to secrets. Database credentials, API keys, admin passwords, and other bits of sensitive information are required for most applications to function properly. You may be tempted to check these credentials into your IaC repository, so that your builds are 100% reproducible. But once they're checked in, they're permanently exposed to anyone with access to your Git repository. If you care about security, it's imperative to avoid this temptation.

The solution is to split the difference: by encrypting all of your secrets, you can safely check them into your repository without fear of exposing them. Then you'll just need access to a single encryption key to "unlock" your IaC repository and have perfectly reproducible infrastructure. Tools like Mozilla's SOPS make this easy. Simply create a single encryption key using Google's or Amazon's key management stores, and any YAML file can be fully encrypted and checked-in to your Git repository.

tl:dr encrypt all your secrets. You'll then only need a single encryption key to unlock your IaC repository.

FINAL THOUGHTS ON KUBERNETES SECURITY

Applications change constantly, and there's no way to ensure that your application code is bulletproof. What Kubernetes does really well is mitigate the severity of attacks and contain splash damage. When someone penetrates your application and makes it through that first layer, they won't get much (or any) farther if you've optimized security settings in accordance with the Kubernetes best practices described here.

With the proper knowhow and attention, a Kubernetes implementation will be more secure and easier to maintain than other systems, specifically because it provides a single platform for everything related to cloud computing. Kubernetes has strong built-in security features, as well as a massive ecosystem of third-party security tooling.

Fairwinds Insights is software tooling that can help ensure cluster security. Insights continuously scans your containers and Kubernetes to pinpoint and prioritize risks, provide remediation guidance and status tracking.

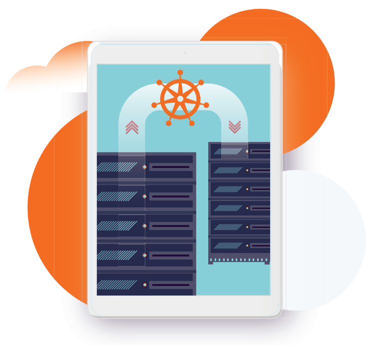
CHECKING FOR KUBERNETES SECURITY BEST PRACTICES, FAIRWINDS INSIGHTS:

- **Automates security at scale** - Fairwinds Insights scans multi-cluster environments against Kubernetes security best practices. Ensure consistent security across multiple teams, cluster and tenancy
- **Integrates shift-left security -** Insights runs security validation checks in your CI/CD pipeline. Configure Insights to show warnings or prevent dev merges based on risky misconfigurations
- **Reduces risk faster** - Insights prioritizes security risks based on severity and category for your team. Use time wisely with a prioritized approach to remediating vulnerabilities

No code is 100% bug-free, and all applications have flaws. Since your applications need to serve traffic to the outside world, it's a matter of if, not when, someone manages to find a hole. Continuously scanning Kubernetes cluster configuration can severely limit the blast radius of an attack. It can make the difference between a minor security incident and a crippling breach.

KUBERNETES BEST PRACTICES FOR COST OPTIMIZATION

Set Just Right CPU and Memory



Kubernetes is a dynamic system that automatically adapts to your workload's resource utilization.

One reason container technology has surpassed the capabilities of traditional virtual machines is its inherent efficiency with regard to infrastructure utilization. Whereas in a traditional virtual machine environment one application is typically run per host, in a containerized environment you can run multiple applications per host, each within its own container. Packing multiple applications per host reduces your overall number of compute instances and thus your infrastructure costs.

Kubernetes is a dynamic system that automatically adapts to your workload's resource utilization. Kubernetes has two levels of scaling. Each individual Kubernetes deployment can be scaled automatically using a Horizontal Pod Autoscaler (HPA), while the cluster at large is scaled using Cluster Autoscaler. HPAs monitor the resource utilization of individual pods within a deployment, and they add or remove pods as necessary to keep resource utilization within specified targets per pod. Cluster Autoscaler, meanwhile, handles scaling of the cluster itself. It watches the resource utilization of the cluster at large and adds or removes nodes to the cluster automatically.

A key feature of Kubernetes that enables both of these scaling actions is the capability to set specific resource requests and limits on your workloads. By setting sensible limits and requests on how much CPU and memory each pod uses, you can maximize the utilization of your infrastructure while ensuring smooth application performance.

To maximize the efficient utilization of your Kubernetes cluster, it is critical to set resource limits and requests correctly. Setting your limits too low on an application will cause problems. For example, if your memory limits are too low, Kubernetes is bound to kill your application for violating its limits. Meanwhile, if you set your limits too high, you're inherently wasting resources by overallocating, which means you will end up with a higher bill.

While Kubernetes best practices dictate that you should always set resource limits and requests on your workloads, it is not always easy to know what values to use for each application. As a result, some teams never set requests or limits at all, while others set them too high during initial testing and then never course correct. The key to ensuring scaling actions work properly is dialing in your resource limits and requests on each pod so workloads run efficiently.

Setting resource limits and requests is key to operating applications on Kubernetes clusters as efficiently and reliably as possible. In this section, I'll take a look at Goldilocks, an open source tool created by Fairwinds that is designed to provide recommendations on resource limits and requests. In addition, I'll touch on some other Kubernetes best practices for optimizing the efficiency of your resource utilization in Kubernetes, including separate instance types and spot instances.

SET KUBERNETES RESOURCES "JUST RIGHT"

The open source project, Goldilocks, by Fairwinds helps teams allocate resources to their Kubernetes deployments and get those resource calibrations just right. Goldilocks is a Kubernetes controller that collects data about running pods and provides recommendations on how to set resource requests and limits. It can help organizations understand resource use, resource costs, and best practices around efficiency of usage.

Goldilocks employs the Kubernetes Vertical Pod Autoscaler (VPA). It takes into account the historical memory and CPU usage of your workloads, along with the current resource usage of your pods, in order to recommend how to set your resource requests and limits. (While the VPA can actually set limits for you, it is often best to use the VPA engine only to provide recommendations.) Essentially, the tool creates a VPA for each deployment in a namespace and then queries that deployment for information.

Teams that are managing multiple clusters may want visibility across their entire environment to undertake cost attribution and resource tuning at scale.

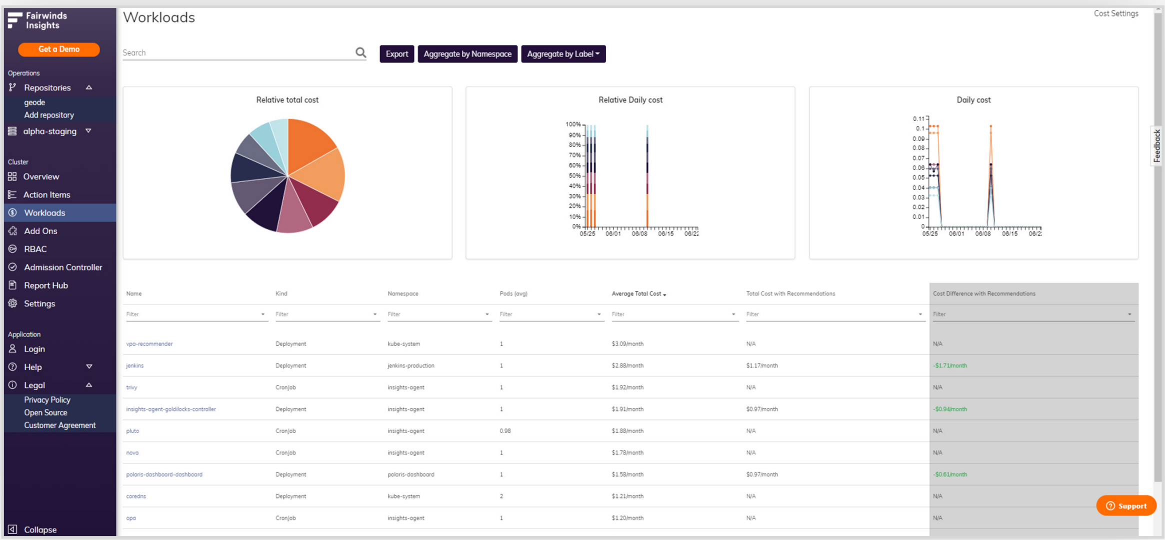
FAIRWINDS INSIGHTS OFFERS RECOMMENDATIONS TO INCREASE EFFICIENCY OF KUBERNETES COMPUTE RESOURCES:

- **Gain visibility** - Dig into application resources and historical usage to discover unknowns. Adjust settings to increase efficiency of Kubernetes
- **Monitor Kubernetes cost** - Evaluate individual applications and find opportunities to reduce costs without impacting application performance
- **Optimize resources** - Insights monitors CPU and memory usage to provide recommendations on resource limits and requests. Maximize the efficiency of CPU and memory utilization for your Kubernetes workloads
- **Allocate cost by namespace or label** - Allocate and group cost estimates by namespace or labeling, making it easier for reports to align to business context

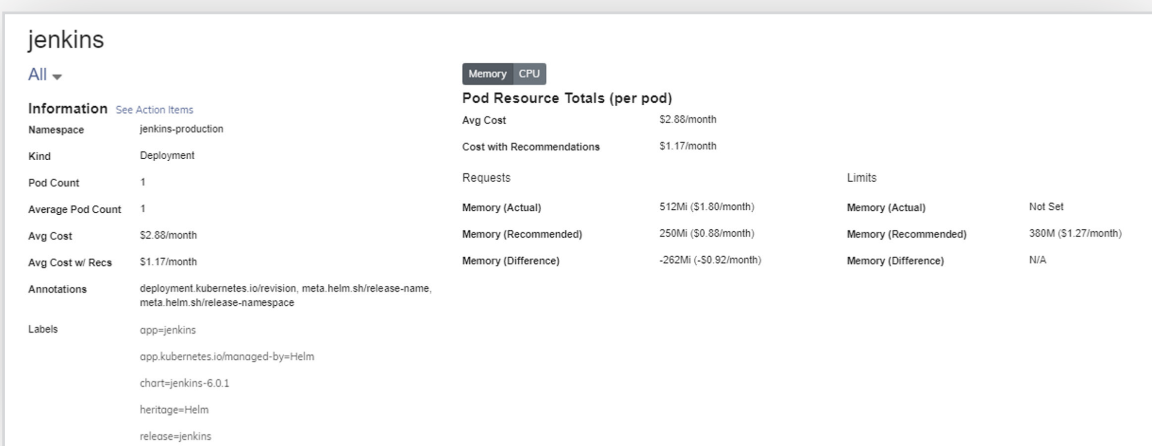
How to Enable Resource Recommendations

Goldilocks is one of the tools Fairwinds Insights deploys to provide workload efficiency and performance optimizations. With Fairwinds Insights, Goldilocks can be deployed across multiple clusters so information is available to teams in a single pane of glass. Fairwinds Insights adds data and recommendations to Goldilocks, including potential cost savings.

The dashboard that appears includes a list of namespaces and deployments with average total cost and cost recommendations.



Many organizations set their CPU and memory requests and limits too high, and when they apply these recommendations they are able to put more pods on fewer Kubernetes worker nodes.



When Cluster Autoscaler is enabled, any extra nodes are removed when they are unused, which saves time and money.

How to Understand the Balance of Your Workloads

Another benefit of Fairwinds Insights and Goldilocks is that it can help you understand if your workloads are CPU-intensive, memory-intensive, or balanced between the two. This data can help you evaluate whether or not you've selected the most efficient workload for your Kubernetes worker nodes. (Here are the machine-type specs for the three largest cloud-providers: AWS, Google Cloud Platform, and Azure).

To view these recommendations, you would have to use kubectl to query every VPA object, which could quickly become tedious for medium-to-large deployments. That's where the dashboard comes in. Once your VPAs are in place, recommendations will appear in the Goldilocks dashboard.

THE DASHBOARD PRESENTS 2 TYPES OF RECOMMENDATIONS DEPENDING ON THE QUALITY OF SERVICE (QOS) CLASS YOU DESIRE FOR YOUR DEPLOYMENTS:

1. **Guaranteed,** which means the application will be scheduled on a node where resources will be assured. In this class, you set your resource requests and limits to exactly the same values, which guarantees that the resources requested by the container will be available to it when it gets scheduled. This QoS class generally lends itself well to the most stable Kubernetes clusters.
2. **Burstable,** which means the application will be guaranteed a minimum level of resources but will receive more if and when available. Essentially, your resource requests are lower than your limits. The scheduler will use the request to place the pod on a node, but then the pod can use more resources up to the limit before it's killed or throttled.

The dashboard provides recommendations for both the Guaranteed and Burstable QoS classes. In the Guaranteed class, consider setting your requests and limits to the VPA "target" field. In general, using this value along with the HPA allows applications to scale.

Note: a third QoS class, BestEffort, means that no requests or limits are set and that the application will be allocated resources only when all other requests are met. Use of BestEffort is not recommended.

Kubernetes Efficiency in Practice

A Goldilocks report showed that a client deployment generated pods that used little CPU but lots of memory. The Kubernetes cluster was set up to use the m5.xlarge AWS instance type, which balances compute, memory, and network resources. We changed some worker nodes to r5.xlarge instances, reducing the size of their Kubernetes cluster and lowering costs substantially.

Specializing Instance Groups for Your Cluster

If you are interested in fine-tuning the instances that your workloads run on, you can use different instance group types and node labels to steer workloads onto specific instance types.

Different business systems often have different-sized resource needs, along with specialized hardware requirements (such as GPUs). The concept of node labels in Kubernetes allows you to put labels onto all of your various nodes. Pods, meanwhile, can be configured to use specific “nodeSelectors” set to match specific node labels, which decide which nodes a pod can be scheduled onto. By utilizing instance groups of different instance types with appropriate labelling, you can mix and match the underlying hardware available from your cloud provider of choice with your workloads in Kubernetes.

If you have different-sized workloads with different requirements, it can make sense strategically and economically to place those workloads on different instance types and use labels to steer your workloads onto those different instance types.

Spot instances (from AWS) and preemptible instances (from Google Cloud) tie into this idea. Most organizations are familiar with paying for instances on demand or on reserved terms over fixed durations. However, if you have workloads that can be interrupted, you may want to consider using spot instances on AWS or preemptible instances on Google Cloud. These instance types allow you to make use of the cloud provider’s leftover capacity at a significant discount—all at the risk of your instance being terminated when the demand for regular on-demand instances rises.

If the risk of random instance termination is something that some of your business workloads can tolerate, you can use the same concept of node labelling to specifically schedule those workloads onto these types of instance groups and gain substantial savings.

FINAL THOUGHTS ON KUBERNETES BEST PRACTICES FOR COST OPTIMIZATION

Setting up and managing clusters and then telling software developers to deploy their apps to those clusters is a complex process. It’s not uncommon for developers to deploy apps but not know how to set the right resource limits or requests. However, using software like Fairwinds Insights or open source tools like Goldilocks empowers developers to remove the guesswork by automating the recommendation process for them. In turn, it opens the door for you to increase the efficiency of your cluster and reduce your cloud spend.

RELIABILITY BEST PRACTICES

Avoid incorrect configuration

Reliability becomes harder and harder to achieve as the business scales. Achieving Kubernetes reliability is complex due to the skill it takes to optimize the capabilities Kubernetes offers.

In a Kubernetes environment, it can be challenging to continue to manage and use technology that predates cloud native applications and Kubernetes, such as Puppet, Chef, Ansible, Packer, and Terraform. Nonetheless, these pre-Kubernetes technologies still often play a role in the Kubernetes world. For example, Terraform can be used to manage infrastructure and Kubernetes dependencies, such as cloud VPCs and networks. Terraform might not be used to manage Kubernetes compute instances or nodes, depending on which tooling is used to provision the cluster. In addition, other providers enable Terraform to manage Kubernetes resources, including Kubernetes manifests (the YAML documents used directly against the Kubernetes API to create various objects inside Kubernetes) and HELM charts.

Many organizations leverage existing Puppet, Chef, or Ansible code to create container images to capitalize on their existing investment. It is natural to want to lift and shift that existing code into the world of containers. However, using these configuration management tools to build Docker images adds unnecessary complexity to your container and Kubernetes efforts, and these tools do not always offer the best, most reliable cloud native experience.

Consider adopting a more direct, more streamlined approach to cloud native applications and infrastructure. Containers abstract and isolate cloud native applications and their dependencies from what's running on the underlying operating system. You can scale these lighter weight containers instead of scaling application server virtual machines. Scale out your application by adding new containers based on the same image vs. waiting for a virtual machine to be created and provisioned by configuration management. While Puppet can be used to create Docker containers, it's more straightforward to create a Docker container in your CI pipeline by maintaining a Docker file. Replace golden machine images with container images as well.

The challenge with using configuration management technology is that it doesn't directly fit into a cloud native container ecosystem. Should you keep doing everything the way you did it in the past and layer Kubernetes on top? The answer is "no." Instead, cloud native methodologies provide an opportunity to adjust how application components communicate and scale.

Reliability becomes harder and harder to achieve as the business scales. Achieving Kubernetes reliability is complex due to the skill it takes to optimize the capabilities Kubernetes offers.

For instance, components of your application use:

- APIs to communicate instead of sharing a common filesystem
- Service discovery to route traffic to services as they scale
- Containers to abstract application dependencies from the underlying operating system.

The more cloud native characteristics an application has, the easier it is to put that application in a container and manage it in Kubernetes. Ensure the reliability of your clusters by shifting to the use of infrastructure as code (IaC).

THE BENEFITS OF INFRASTRUCTURE AS CODE

Simply put, IaC is the process of managing your IT infrastructure using configuration files.

HERE WE DISCUSS IAC ADVANTAGES INCLUDING:

- Reduced human error and future proofing
- Repeatability and consistency
- Disaster recovery
- Improved auditability

Reduced Human Error and Future Proofing

IaC and automation reduce human error by creating predictable results. You can produce new environments to test infrastructure upgrades to validate changes without impacting production. If you want to apply changes to infrastructure across multiple environments, using code reduces errors because focus and attention to detail are less impacted by repetitive manual work.

IaC also helps to reduce single points of failure against talent loss or tech progress by documenting the infrastructure. In other words, the code and comments increase awareness about the design and configuration of infrastructure. They help with training as well, reducing the need for subject matter experts to get developers up to speed.

Repeatability and Consistency

The repeatability of IaC helps you create consistent infrastructure in other regions much more rapidly. This feature frees up time to move on to the next set of problems, such as how to route traffic to applications throughout the region and how to test failover without impacts on production.

Disaster Recovery

How long does it take to rebuild a container image in an emergency (e.g., deploy new code to address an application outage or degradation)? If you're using manual processes or complex chains of tooling, then that disaster recovery (DR) process will take longer. The reliability of an application is impacted by the ability to pivot and the speed to redeploy. Be sure you know what that process looks like and how to put in place the right practice, tooling, and underlying processes to make a Kubernetes deployment as straightforward as possible.

Improved Auditability

IaC also helps track changes to an audit infrastructure. Because your infrastructure is represented in code, commits to your Git repository reflect who, when, and why changes were made. You'll be able to look at the code and know how environments were built, what's happening, and why.

KUBERNETES RELIABILITY BEST PRACTICES

BELOW, WE WILL HIGHLIGHT THE FOLLOWING KEY KUBERNETES BEST PRACTICES RELATED TO RELIABILITY:

- Simplicity vs. complexity
- High-availability (HA) architecture/fault tolerance
- Resource limits and autoscaling
- Liveness and readiness probes

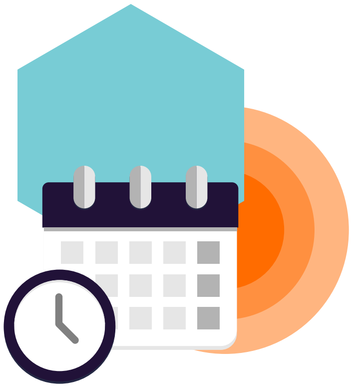
Simplicity vs. Complexity

Unfortunately you can introduce too much complexity into your Kubernetes environments.

AVOID COMPLEXITY BY KEEPING IT SIMPLE. HERE ARE THREE WAYS TO DO THAT:

1. **Service delivery vs. traffic routing** - Manually maintained DNS entries can be used to point to an application, and DNS hostnames can be hardcoded into application components so they can communicate. However, rather than using traffic routing, instead use service delivery, which is a more streamlined, dynamic solution. Service delivery enables a user or another application to find instances, pods, or containers. Service delivery is required because your application is scaling in and out, and changes are happening at a fast rate.
2. **Application configuration** - There is a shift from Puppet Hiera, Chef Data Bags and Ansible Variables to files or environment variables in your container. Those are populated by Kubernetes ConfigMaps or Secrets. You can run an application in multiple environments, but the configuration will differ because you have different ConfigMaps or Secrets in Kubernetes for each environment.
3. **Configuration management tools** - teams often use configuration management tools like Puppet to continuously correct state in a virtual machine (VM) that runs an application, which offers some auditability and security. For example, if someone connects to a VM that runs an application and changes a config file, Puppet will change it back. Containers, however, are more ephemeral. If you need to change something about how an application runs, CI/CD best practices dictate that you should build and then deploy a new container image through your CI pipeline instead of attempting to modify an existing container.

HA Architecture/Fault Tolerance



Kubernetes helps **improve reliability** by making it possible to schedule containers across multiple nodes and multiple availability zones (AZs) in the cloud.

Kubernetes helps improve reliability by making it possible to schedule containers across multiple nodes and multiple availability zones (AZs) in the cloud. Anti-affinity allows you to constrain which nodes in your pod are eligible to be scheduled based on labels on pods that are already running on the node rather than based on labels on nodes. With node selection, the node must have each of the indicated key-value pairs as labels for the pod to be eligible to run on a node. When you create a Kubernetes deployment, use anti-affinity or node selection to help spread your applications across the Kubernetes cluster for high availability.

Kubernetes HA means having no single point of failure in a Kubernetes component. An example of a component might be a Kubernetes API server or the etcd database where state is stored in Kubernetes. How do you help ensure these components are HA? Let's say you are using Kubernetes on premises and you have three master servers with a load balancer that runs on a single machine. While you have multiple masters, your one load balancer is a single point of failure for the Kubernetes API. You need to avoid this.

If a redundant component in your Kubernetes cluster is lost, the cluster keeps operating because K8S best practice is to deploy a number of redundant instances based on the component (etcd odd number, 3+, API servier 2+, kube-scheduler 2+ for example). If you lose a second component, then what happens? If you have three masters and you lose one, the two remaining masters could get overloaded, contributing to the degradation or potential loss of another master. It's key to plan the resiliency of your cluster according to the risk your business can tolerate.

tl:dr plan your fault tolerance stra1egy and employ HA redundancy based on your workload.

Resource Limits and Autoscaling

Resource requests and limits for CPU and memory are at the heart of what allows the Kubernetes scheduler to do its job well. If a single pod is allowed to consume all of the node CPU and memory, then other pods will be starved for resources. Setting limits on what a pod can consume increases reliability by keeping pods from consuming all of the available resources on a node (this is referred to as the "noisy neighbor problem").

Autoscaling, in turn, can increase cluster reliability by allowing the cluster to respond to changes in load. Horizontal Pod Autoscaler (HPA) and cluster autoscaling work together to provide a stable cluster by scaling your application pods and cluster nodes.

Reliability first requires good resource requests and limits, and the Cluster Autoscaler will have a hard time doing its job if your resource requests are not set correctly. The Cluster Autoscaler relies on the scheduler to know that a pod won't fit on the current nodes, and it also relies on the resource request to determine whether adding a new node will allow the pod to run.

tl:dr set limits on what a pod can consume to increase reliability. This avoids the noisy neighbor problem.

Liveness and Readiness Probes

Another important facet of cluster reliability involves the concept of "self-healing." The idea here is to automatically detect issues in the cluster and automatically fix those issues. This concept is built into Kubernetes in the form of liveness and readiness probes.

A liveness probe indicates whether or not the container is running or alive, and it is fundamental to the proper functioning of a Kubernetes cluster. If this probe is moved into a failing state, then Kubernetes will automatically send a signal to kill the pod to which the container belongs. In addition, if each container in the pod does not have a liveness probe, then a faulty or non-functioning pod will continue to run indefinitely, using up valuable resources and possibly causing application errors.

A readiness probe, on the other hand, is used to indicate when a container is ready to serve traffic. If the pod is behind a Kubernetes service, the pod will not be added to the list of available endpoints in that service until all of the containers in that pod are marked as ready. This procedure allows you to keep unhealthy pods from serving any traffic or accepting any requests, thus preventing your application from exposing errors.

Both probes check that the Kubernetes cluster performs on your containers at set intervals. Each probe has two states, pass and fail, along with a threshold for how many times the probe has to fail or succeed before the state is changed. When configured correctly on all of your containers, these two probe types provide the cluster with the ability to “self-heal.” Problems that arise in containers will be automatically detected, and pods will be killed or taken out of service automatically.

tl:dr configure liveness probes and readiness probes to provide your cluster with the ability to self-heal.

FINAL THOUGHTS ON BUILDING RELIABLE KUBERNETES CLUSTERS

Reliability in a Kubernetes environment is synonymous with stability, streamlined development and operations, and a better user experience. In a Kubernetes environment, reliability becomes much easier to achieve with the right configuration. Many factors need to be considered when building a stable and reliable Kubernetes cluster, including the possible need for application changes and changes to cluster configuration. Steps include setting resource requests and limits, autoscaling pods using a metric that represents application load, and using liveness and readiness probes.

Reliability becomes much easier to achieve with the right configurations. With multiple teams working across multiple clusters, Fairwinds Insights provides visibility into your Kubernetes security configurations. The software continuously scans clusters to identify image, container, cluster and Kubernetes misconfigurations. It tightly integrates into your CI/CD pipeline checking for misconfigurations against your security and compliance and reliability policies. Identify risk early. Fix fast. Gain peace of mind that your application will run reliably.

POLICY ENFORCEMENT BEST PRACTICES

Avoid Consistency Multi-User, Cluster, Tenant Kubernetes Environments

Managing cluster configuration becomes unwieldy fast as workloads are inconsistently or manually deployed and modified.

In most cases, organizations pilot Kubernetes with a single application. Once successful, these organizations commit to Kubernetes across multiple apps, development and ops teams.

Managing cluster configuration becomes unwieldy fast as workloads are inconsistently or manually deployed and modified. Without guardrails, there are likely to be discrepancies in configurations across containers and clusters which can be challenging to identify, correct and keep consistent.

Manually identifying these misconfigurations is highly error-prone and can quickly overwhelm ops teams with code review. When managing multi-cluster environments with a team of engineers, creating consistency requires you to establish Kubernetes policies to enforce security, efficiency and reliability.

POLICIES FALL INTO THREE CATEGORIES:

1. **Standard policies** - enable best practices across all organizations, teams and clusters. Examples include disallowing resources in the default namespace, requiring resource limits to be set, or preventing workloads from running as root
2. **Organization-specific policies** - enforce best practices that are specific to your organization. Examples include requiring particular labels on each workload, enforcing a list of allowed image registries, or policies that help with compliance and auditing requirements
3. **Environment-specific policies** - enforce or relax policies for particular clusters or namespaces. Examples include stricter security enforcement in prod clusters, or looser enforcement in namespaces that run low-level infrastructure

Simply putting in place a best practices document for your engineering team doesn't work - it will be likely forgotten or ignored. Kubernetes policy enforcement helps prevent common misconfigurations from being deployed into the cluster, enables IT compliance and governance and allows teams to ship with confidence knowing that guardrails are in place.

When does Policy Enforcement Make Sense

- **Shared Cluster:** You're a Platform Engineering/ Operations team building or running a Shared Kubernetes Cluster that serves multiple app teams.
- **Multi-Cluster:** You're a Platform Engineering/Operations team running multiple clusters, with plans to expand your footprint in the cloud, on-prem, or both.
- **Service Ownership:** Development teams own all things application and operations (e.g., Service Ownership philosophy) and want to help engineers avoid mistakes that distract from building their app.

KUBERNETES POLICY ENFORCEMENT OPTIONS

There are three options you can take when approaching Kubernetes policy enforcement.

Develop internal tools

Of course engineers like to develop their own tools for a problem, however, here leaders need to decide whether their team can spend the time, money and resources developing and maintaining home-grown tooling, rather than working on problems that are specific to their business.

Deploy open source

There are a number of different open source tools that can help with security, reliability and efficiency configuration. There are open source auditing tools for container scanning and network sniffing, as well as Fairwinds' own contributions, like Polaris, Goldilocks, Nova and Pluto that can audit Kubernetes clusters.

POLARIS COMES WITH 20 BUILT-IN CHECKS AROUND SECURITY, EFFICIENCY AND RELIABILITY. SOME EXAMPLE CHECKS POLARIS LOOKS OUT FOR INCLUDES:

- Whether a readiness or liveness probe is configured for a pod
- When an image tag is either not specified or set to latest
- When the `hostNetwork` or `hostPort` attribute is configured
- When memory and CPU requests and limits are not configured
- When `securityContext.privileged` is true or when `securityContext.readOnlyRootFilesystem` is not true (amongst a number of other security configuration checks)

The Kubernetes community has settled on a powerful open standard for creating configuration policies: Open Policy Agent, or OPA. Open Policy Agent (OPA) is an open source, general-purpose policy engine that unifies policy enforcement across the stack. It provides a high-level declarative language that allows you to specify policy as code and use simple APIs to offload policy decision-making from your software. OPA enables you to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more.

Benefits of OPA: Policy as Code

OPA allows users to set policies across infrastructure and applications.

Some examples include:

- Which registries images can be downloaded from
- Which OS capabilities a container can execute with
- Which namespaces are allowed to run sensitive workloads
- Labels that must be specified for certain resources
- Disallowing deprecated or dangerous resource types
- Enforcing naming schemes or internal standards

OPA is also context-aware, so administrators can make policy decisions based on what's happening - who is making the change, which cluster or namespace is affected, etc.

As explained by one of its founders, "OPA gives policy its own lifecycle and toolsets, so policy can be managed separately from the underlying systems that the policy applies to...OPA provides local enforcement for the sake of higher availability, better performance, greater flexibility, and more expressiveness than hard-coded service logic or ad-hoc domain-specific languages."

If you select the open source route, your team will spend time deploying and managing each tool. You'll need to ask whether your team has the bandwidth for this and if it will enable you to focus on the apps or services that make you money.

Security and Compliance Software

Here you have the software expense, but your team can immediately take action by fixing inconsistencies and enforcing policy throughout your entire CI/CD pipeline.

To address the challenges around policy-enforcement in Kubernetes, Fairwinds Insights can help enforce policies to improve security, efficiency and reliability so that clusters are secure, scale properly to avoid downtime and costs are kept in control.

THE BENEFITS OF USING FAIRWINDS INSIGHTS VS. USING STANDALONE OPEN SOURCE OPTIONS INCLUDE:

- A single solution that includes CI, Admission Controller, and In-Cluster integrations across your organization
- Write once, use everywhere - policies can be configured once and deployed into as part of CI/CD, Admission controller, and In-Cluster checks
- Single platform for managing results across multiple clusters, pushing notifications, creating tickets, etc.

OPA and Fairwinds Insights

With OPA, Kubernetes users benefit from policy as code where policies are codified and enforcement is automated. A SaaS platform like Fairwinds Insights can provide a much more seamless, unified experience for managing OPA at scale.

OPA IS INTEGRATED INTO FAIRWINDS INSIGHTS IN THREE MAJOR WAYS:

1. As a CI/CD hook, auditing Infrastructure-as-Code as part of the code review process
2. As an Admission Controller (aka Validating Webhook), which will stop problematic resources from entering the cluster
3. As an in-cluster agent, repeatedly scanning for problematic resources

Fairwinds Insights can take the same OPA policies and federate them out to all three contexts, and to as many clusters as you'd like.

FINAL THOUGHTS ON KUBERNETES POLICY ENFORCEMENT BEST PRACTICES

When running multi-tenant or multi-cluster environments with many teams and users, Kubernetes Policy Enforcement is essential.

USING A TOOL LIKE FAIRWINDS INSIGHTS TO ENFORCE YOUR POLICY-AS-CODE WILL OFFER THESE BENEFITS:

- **Enforce consistency** - Automate deployment guardrails and security best practices through Open Policy Agent (OPA) integrations at the CI/CD stage, or as an admission controller
- **Prevent mistakes** - Automate issue detection during application development to prevent mistakes from entering production in the first place
- **Improve security** - Gain continuous visibility into your Kubernetes security posture by auditing workloads for misconfigurations and weaknesses
- **Reduce cost** - Increase the efficiency of Kubernetes resource usage to save you money in the cloud or capacity in the data center
- **Save time** - Eliminate the guesswork and increase speed-to-market with built-in collaboration tools, notifications, workflows, and integrations into the tools that teams use everyday

MONITORING & ALERT BEST PRACTICES

Optimizing Monitoring and Alerting in Kubernetes

Reliability becomes harder and harder to achieve as the business scales. Achieving Kubernetes reliability is complex due to the skill it takes to optimize the capabilities Kubernetes offers.

Organizations commonly want availability insurance, yet few organizations monitor their environments well. For one thing, environments are always changing, and it's hard for monitoring to keep up. For another, monitoring configuration is often an afterthought—it isn't set up until a problem occurs, and monitoring updates are seldom made as workloads change. When the average organization finally recognizes its need for application/system monitoring, the team is too overwhelmed just trying to keep infrastructure and applications "up" to have the capacity to look out for issues.

When load is high, will the tools responsible for the infrastructure even notice? Will they automatically scale? If not, will the right people be alerted? Even monitoring the right things to identify the problems the application or infrastructure is facing on a day-to-day basis is beyond the reach of many organizations.

The truth is that monitoring done right is still just a pipedream for most companies. It's a serious problem. Insufficient monitoring can interrupt operations. Lack of visibility into your systems can impact site reliability engineering (SRE) teams as well because they cannot respond to issues as fast as needed or respond to the right issues because they are monitoring the wrong things. Insufficient monitoring introduces a lot of toil because you need to constantly monitor systems to ensure they reflect the state that you want.

In a Kubernetes environment, an additional challenge involves monitoring management that reflects the state of clusters and workloads. Existing tools rely on manual configuration of monitoring state, which introduces toil to SRE teams. Manual configuration also increases availability and performance risks because monitors may not be present or accurate enough to trigger changes in key performance indicators (KPIs). Service-level agreement (SLA) breaches can result due to undetected issues, and noisy pagers can result due to incorrect monitor settings.

One solution is the push towards observability—the ability to discover unknown unknowns. Kubernetes best practices involve recognition that monitoring is key and requires the use of the right tools to optimize your monitoring capabilities.

WHAT NEEDS MONITORING AND WHY?

Here, we focus on what kinds of things need to be monitored and why, as well as introduce tools that aggregate data to be operationalized.

WE WILL HIGHLIGHT THE FOLLOWING KUBERNETES BEST PRACTICES RELATED TO MONITORING AND ALERTING:

- Centralized aggregation, management, and monitoring standards
- Monitoring as code
- Infrastructure vs. workload ownership

Centralized Aggregation, Management, and Monitoring Standards

Monitoring has always been challenging, and the problem is magnified in a dynamic, ever-changing Kubernetes environment. With Kubernetes, you have to build monitoring systems and tooling to respond to the dynamic nature of the environment. Thus, you will want to focus on availability and workload performance. One typical approach is to collect all of the metrics you can and then use those metrics to try to solve any problem that occurs. It makes the operators' jobs more complex because they need to sift through an excess of information to find the information they really need. At present, there's also not a de facto standard for what to alert on.

In a world where infinite options for tooling seem to many like a birthright, standardization may seem all but impossible. However, standardization has already happened in a number of places. Linux, for example, has established best practices because it won the production operating system wars.

In response to these variations and to widespread monitoring challenges, monitoring standards have emerged in the open source community. Each monitoring system may have a different way to collect and display metrics, which is why Prometheus, an open source service monitoring system, offers a standardized way to collect and expose metrics in clusters and your workloads. OpenMetrics is an effort by the community to create an open standard for transmitting those metrics. Both are attempting to standardize on these monitoring tools and the format of all of the associated metrics.

Key Monitoring Alerts

Monitors often created as a best practice include the following:

- Kubernetes deployment with no replicas
- Horizontal Pod Autoscaler (HPA) scaling issues
- Host disk usage
- High IO wait times
- Increased network errors
- Increase in pods crashed
- Unhealthy Kubelets
- nginx config reload failures
- Nodes that are not ready
- Large number of pods that are not in a Running state
- External-DNS errors registering records

Datadog is one of the predominant software as a service (SaaS) tools that can handle the monitoring for you. Datadog has already built out the systems and mechanisms for you to start monitoring on day one. It's costly, but out-of-the-box you get a robust monitoring solution that does not require a lot of implementation effort. You can use Datadog to create metrics and then use their portal to view and manage monitors.

Infrastructure vs. Workload Ownership

Monitoring is primarily set up to focus on infrastructure. But organizations are also concerned with how to monitor cluster workloads.

Because a diverse set of stakeholders is involved in the process, you must determine who is responsible for what from both an infrastructure and a workload standpoint. For instance, you want to make sure the right people are alerted at the right time to limit the noise of being alerted about things that do not pertain to you.

Tooling must be flexible enough to meet complex demands, yet easy enough to set up quickly so that we can move beyond tier 1 monitoring (e.g., Is it even working?"). Tier 2 monitoring requires dashboards that reveal where security vulnerabilities are, whether or not compliance standards are being met, and targeted ways to improve.

Impact and urgency are key criteria that must be identified and assessed on an ongoing basis. Regarding impact, it is critical to be able to determine if an alert is actionable, the severity based on impact, and the number of users or business services that are or will be affected. Urgency also comes into play. For example, does the problem need to be fixed right now, in the next hour, or in the next day?

Monitoring as Code

In parallel with IaC, it is important to implement monitoring as code. The next evolution in monitoring, Astro is an open source software project built by Fairwinds to help you achieve better productivity and cluster performance. Astro was built to work with Datadog. Astro watches objects in your cluster for defined patterns and manages Datadog monitors based on this state. As a controller that runs in a Kubernetes cluster, it subscribes to updates within the cluster. If a new Kubernetes deployment or other objects are created in a cluster, Astro knows about it and creates monitors based on that state in your cluster. Essentially, it provides a mechanism for dynamically creating and managing alerts in a way that Datadog can understand.

All you have to do is determine what you want to monitor and describe the types of monitors you want for different objects in a cluster. From there, Astro manages that state based on your Kubernetes cluster. Once you define how you want Astro to manage that state, monitoring is done by annotating objects in your cluster. As long as the monitors have the right annotations, they will automatically be created and routed to the right notification endpoints.

Managing Datadog Monitors in a Kubernetes Deployment

Expanding on monitoring management features, Astro offers a reliable system of checks and balances, and it can be used to keep your monitors in sync, prevent outages due to misconfigured monitors, and fight pager fatigue.

THE TOOL PROVIDES THREE KEY ELEMENTS TO GREATLY SIMPLIFY DATADOG MONITOR MANAGEMENT:

1. **Automated lifecycle management of Datadog monitors.** Given configuration parameters, the utility will automatically manage defined monitors for all relevant objects within the Kubernetes cluster. As objects change, monitors are updated to reflect that state, eliminating the need for manual configuration.
2. **Correlation between logically bound objects.** Astro can manage monitors for all objects within a given namespace, which ensures greater consistency across monitor configurations.
3. **Templating values from Kubernetes objects into managed monitors.** Any data from a managed Kubernetes object can be inserted into a managed monitor, which produces more informative and contextualized alerts and makes it easier to triage issues.

Astro takes advantage of new Kubernetes-native patterns to help engineers declaratively manage Datadog monitors with the agility to match the ephemeral nature of these environments. Without Astro, the engineering team would have to manually create those monitors, a time-consuming and error-prone process. Now, every time a Kubernetes deployment is created with the Astro/owner annotation, Astro automatically creates a Datadog monitor for it and provides an alert if it goes down.

While Datadog already offers visibility into the health and performance of a Kubernetes environment, when you add the Fairwinds Insights integration to your Datadog deployment, it adds focus on your Kubernetes cluster and helps you identify issues before they turn into an alarm going off in your Datadog dashboard.

Insights integrates into the CI/CD pipeline, identifying security, efficiency, and reliability misconfigurations before they get to your production environment. Insights can also apply Kubernetes best practice policies and enforce policies for compliance, helping you build and maintain a more secure, efficient, and reliable Kubernetes deployment even before you move your applications live and begin monitoring with Datadog.

Adding Fairwinds Insights to the mix helps you find out which configurations are risky or wasting resources. By integrating Insights into Datadog, you can find and prevent security and compliance risk in pre-production and identify misconfigurations that could be breaking your build. So the Insights integration acts more like a home inspector — it can't prevent the fire alarm from going off, but it can reduce the likelihood that it will by resolving issues earlier in the process.

FINAL THOUGHTS ON OPTIMIZING KUBERNETES MONITORING AND ALERTING

It is difficult to always know what to monitor ahead of time, so organizations need at least enough context to figure out what's going wrong when someone inevitably gets woken up in the middle of the night and needs to bring everything back online. Without this level of understanding, engineering teams cannot parse what should be monitored and know when to grin and bear turning on an alert.

Monitoring the health of infrastructure and applications is essential to every company—even if they're doing it poorly—even if, as in many organizations, it is for the sole purpose of understanding the root cause of problems.

Today, cloud infrastructure is still in its infancy, but organizations are standardizing around tools like Kubernetes. As specific tools become more and more universally accepted as standards, the focus can shift from innovation velocity toward hardening. Once they get to the hardening stage, organizations can then set standards for monitoring. These standards make it easier to monitor more of the right things. In the not-so-distant future, streamlined configurability and accessibility, along with straightforward tier 2 monitoring, will be the norm instead of the exception.

Fairwinds provides software built in response to years of Kubernetes managed services.

CONCLUSION

Kubernetes security, efficiency, reliability, and monitoring and alerting are complex to say the least. Kubernetes offers a rich capability set that requires your deployments to work well at both the deployment level and the cluster level. Understandably, it can be hard to know where to start.

Kubernetes enables organizations to derive maximum utility and productivity from containers and build cloud native applications that can run anywhere. To take full advantage of Kubernetes, the key is to ensure your Kubernetes implementation is as bulletproof as possible.

HELPFUL RESOURCES:

- Polaris helps engineers align their Kubernetes deployment manifests with best practices, detecting issues related to security, networking, and container images.
- Goldilocks saves engineering time by recommending resource requests and limits (essentially, data-informed CPU and memory settings) for Kubernetes deployments. Both of these tools work nicely at the handoff from development to production, providing developers with a critical feedback loop before they release.
- Fairwinds Insights is software for DevSecOps managing multiple clusters and teams that want the benefit of visibility across their Kubernetes environment. Insights continuously scans your environment for misconfigurations against security and compliance, policy and cost optimization.

BENEFITS OF FAIRWINDS SECURITY, COST OPTIMIZATION AND POLICY ENFORCEMENT

Fairwinds provides software built in response to years of Kubernetes managed services. Our in-house Kubernetes experts have seen every kind of cluster implemented in every possible way. We know what to look for, and this depth and breadth of hard-won experience allows us to anticipate and address the range of challenges that can occur. We have seen everything that can go wrong, and we know how to make things go right.

Fairwinds Insights was purpose-built to address problems we saw with Kubernetes security, efficiency and reliability. We operationalized leading open source tools including Polaris, Goldilocks and Trivy to help with the overhead associated with installing, maintaining and updating disparate open source tools across multiple clusters.

WHY FAIRWINDS

Fairwinds is your trusted partner for Kubernetes security, policy and governance. With Fairwinds, customers ship cloud-native applications faster, more cost effectively, and with less risk. We provide a unified view between dev, sec, and ops, removing friction between those teams with software that simplifies complexity. Fairwinds Insights is built on Kubernetes expertise and integrates our leading open source tools to help you save time, reduce risk, and deploy with confidence.

WWW.FAIRWINDS.COM