Google Cloud

Kubernetes to Cloud Native

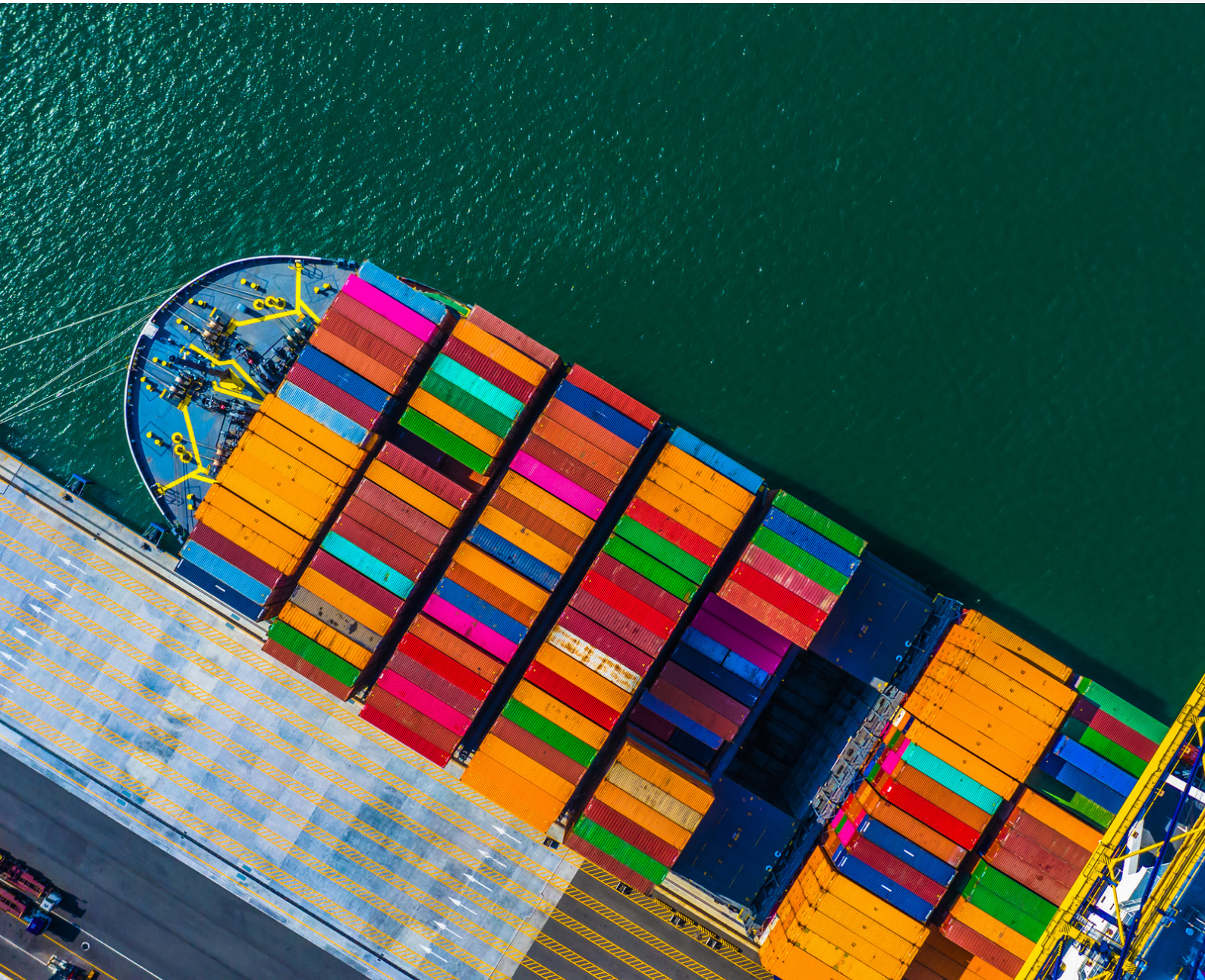
Jumpstart your Journey Developing on GKE

End-to-End Developer Journey on GKE Ebook

Google Cloud

Table of contents

Google Cloud



Kick-start your journey developing on GKE

Your journey with Kubernetes starts here with Google Kubernetes Engine (GKE). GKE is the most mature container orchestration service today, delivering a fully managed service and hands-off experience with the Autopilot mode of operation. GKE provides industry-first capabilities such as release channels, multi-cluster support, unique four-way auto scaling, node auto repair, and can support up to 15K nodes in a cluster. GKE's built-in integrations for developer toolchains, CI/CD, logging, and monitoring allow for an easier end-to-end developer experience.

Our modern end-to-end platform is built on cloud-native principles you are already familiar with and prioritizes speed, security, and flexibility, in ways that are highly differentiated from other cloud platforms.

This e-book will serve as a reference guide for you as you begin your journey developing on GKE, covering everything from code to build to run to operate and manage. Even if it's your first time using GKE, this will be a valuable resource highlighting important considerations and best practices. By implementing the technical recommendations, following the steps, and utilizing the tools described in this book, you can reach the following goals:

1

Write, deploy, and debug code faster with Cloud Code and Cloud Shell

2

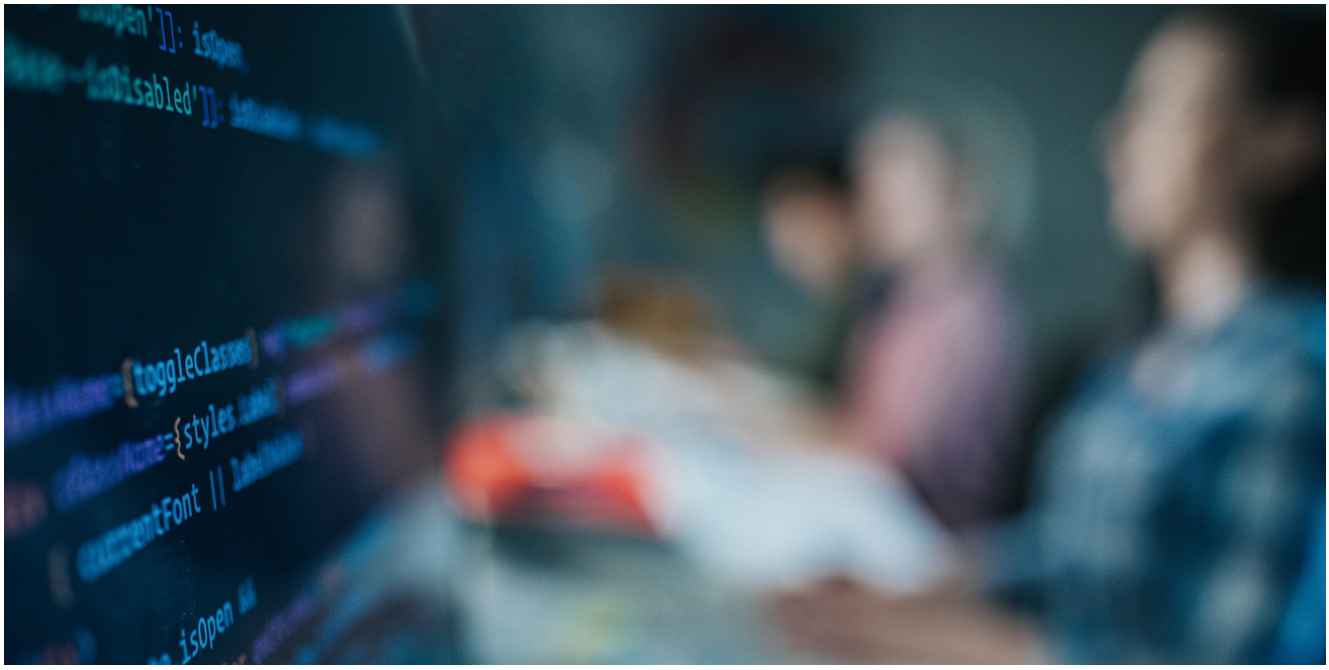
Continuously integrate and deliver updates with Cloud Build

3

Run easily, securly, cost effectively at scale with GKE

4

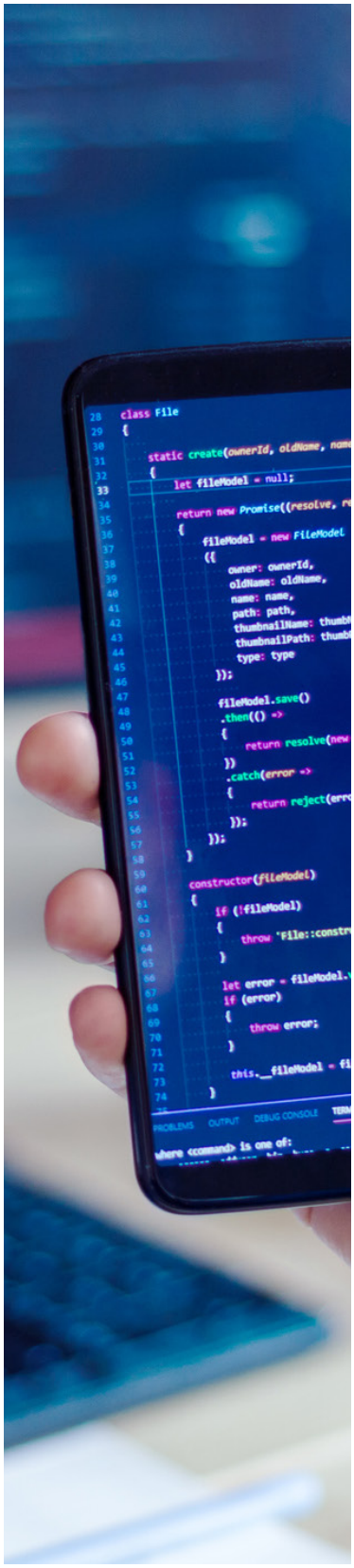
Debug and troubleshoot with Google Cloud's operations suite



CODE

Getting started with Kubernetes developer tools

Starting a coding project can be extremely daunting, especially if you don't have the right tools. Developer tools that keep you focused on code, instead of managing infrastructure, are crucial in kick-starting your development journey. Begin your coding on GKE with tools that are designed to help you develop more efficiently, enabling you to write, deploy and debug faster, all within a familiar integrated development environment (IDE).



Get started with Kubernetes app development for free with Cloud Shell

As you get started developing with GKE, here are a few things to keep in mind. With Cloud Shell anyone with a Google account can get access to a cloud-based IDE that includes the following:

Cloud Code: an IDE plug-in that helps developers stay productive when developing cloud-native applications

Minikube: a tool that makes it easy to run a single-node Kubernetes cluster in Cloud Shell

Cloud Build Local Builder: a tool for running your continuous integration process

You can jump right in by clicking the button.

This tutorial will walk you through how to run and build a Kubernetes application.

Develop with Kubernetes where you are most comfortable

Use Cloud Code for local development

If you're trying to run your Kubernetes application on your local machine, Cloud Code makes it easy to deploy to Docker Desktop. Or, it can create and start a minikube cluster for you. Minikube runs a single-node Kubernetes cluster on your local machine.

Cloud Code can also work with remote clusters to offload CPU and memory resources from your laptop. This works whether the Kubernetes cluster is managed by GKE or a conformant cluster.

Reduce context switching: Cloud Code provides tools built right into your IDE

Developers are most productive when writing code within the environments with which they are already comfortable. Cloud Code's built-in tools streamline developer workflows significantly. Moreover, context switching can not only be time consuming, but also can easily break a developers flow. Cloud Code reduces this context switching by including hands-off hot-code reloading so you can see your changes update live in your development system. It also comes with the integrated Kubernetes Explorer to help you view and debug your application resources.

Kubernetes development is more accessible and painless with Cloud Code's built-in log viewer. It makes viewing logs for each revision simpler by giving you access to revision and service logs with just one click.

Get help with your cloud-native configuration and tools

Cloud Code lets you spend more time writing code and less time configuring your application, with authoring support features such as inline documentation, completions, and schema validation, a.k.a., “linting.” As Kubernetes has grown more popular, many developers have extended the Kubernetes API with new Operators and Custom Resource Definitions (CRDs). These new Operators and CRDs expanded the Kubernetes ecosystem with new functionality such as continuous integration and delivery, machine learning, and network security. Cloud Code supports over 400 Kubernetes CRDs out of the box, while providing authoring support to make it easier to write, understand, and see errors in the YAML for a wide range of Kubernetes CRDs. Cloud Code’s enhanced authoring support lets you leverage this custom Kubernetes functionality by creating a resource file that conforms to the CRD.

As a developer, you want to focus on writing code, not containerizing it. Buildpacks is an open-source technology that makes it fast and easy for you to create secure, production-ready container images from source code — no Dockerfile necessary.

Iterate quickly with hot-reloading of your cloud-native apps

Developers creating Kubernetes-native applications spend a lot of time building and managing container images across registries, manually updating their Kubernetes manifests, and redeploying their applications every time they make even the smallest code change. Cloud Code and Skaffold are your keys to automate these repetitive tasks.

Skaffold simplifies common operational tasks that you perform when developing on Kubernetes, letting you focus on your code changes and seeing them rapidly reflected on your cluster. Enabling Skaffold Watch Mode watches local source code for changes, and rebuilds and redeploys applications to your cluster in real time.

Cloud Code also handles forwarding the ports from your Kubernetes application onto your local machine so you can easily view and interact with your UIs and APIs with local browsers and clients.

Debug in real time

Fixing bugs quickly and efficiently can often be difficult during Kubernetes development. Cloud Code lets you debug in real time, saving you valuable time when developing and allowing you to debug your Kubernetes services the same way you do locally. With Cloud Code, you can simply place breakpoints in your code. Once you trigger that code path in your app, you can step through the code, hover over variable properties, and view the logs from your container.

BUILD

Continuous integration and continuous delivery (CI/CD)

When a large number of developers work on related systems, coordinating code updates can be a hard problem, and changes from different developers can be incompatible. The practice of continuous integration (CI) allows developers to integrate all their code changes back into a main branch as often as possible, exposing failures faster by revealing issues as early as possible in the process. The CI pipeline includes steps to validate changes such as linting, testing, and building.

A CI pipeline typically produces an artifact that you can deploy in later stages of the deployment process with continuous delivery (CD). CD lets you release code at any time. CD operates on the artifact produced by CI pipelines. CD pipelines can run for much longer than CI pipelines, especially if you're using more elaborate deployment strategies.

To make this clearer, it's useful to look at a concrete example. Figure 1 shows how a CI/CD pipeline might look using Google tools for containers running on GKE.

Figure 1: A CI/CD pipeline has several steps, from writing code to deploying a new container

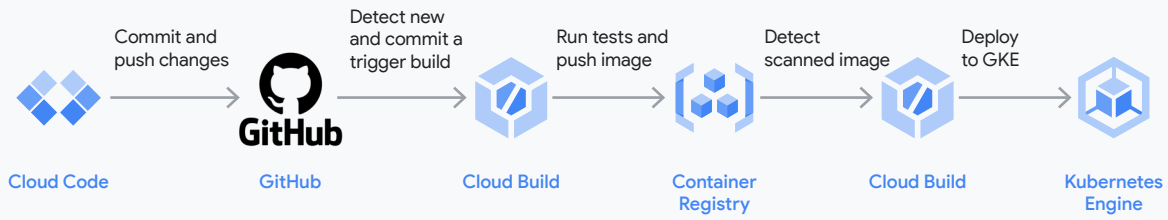
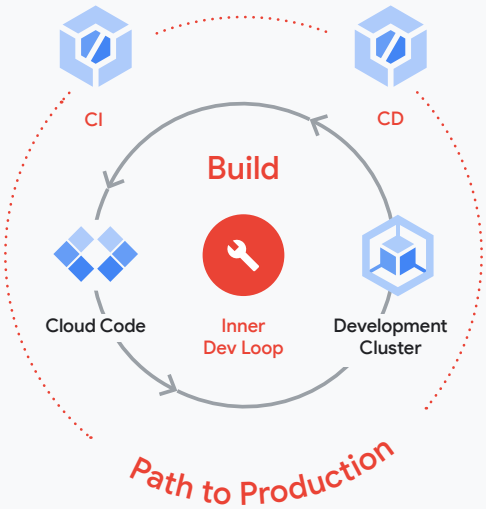


Figure 2:
Local and remote development loops



Create pipelines that enable rapid iteration

01

The time between when a developer makes a code change and when you have a running version of the application should be as short as possible. Speed is especially important when developing feature branches that developers are iterating on quickly. By implementing an automated build process along with a suite of automated unit tests that run in a few minutes or less, you can gain confidence that your software works as expected.

Cloud Build, Google Cloud's CI/CD platform, can help drive fast feedback in a number of ways. First, you can trigger CI on every branch. Working in small batches also ensures that developers get regular feedback on the impact of their work. Secondly, because Cloud Build lets you run multiple concurrent builds per project, you can optimize the speed of builds and in turn developer productivity. Third, you can choose from four high-CPU virtual machine types to increase the speed of your builds. And finally, Cloud Build lets you caches container build artifacts with Kaniko. By storing and indexing intermediate layers within a container image registry such as Google's own Container Registry, Kaniko makes them available for use by subsequent builds. For more information, see Using Kaniko cache.

Treat everything as code

02

Treat everything as code: A core principle of CI/CD is that the source of truth for everything about an application should be represented as text in a source repo. This means that the application itself, documentation, runtime policy and config — and critically, the CI/CD pipeline that delivers the application — are all changed and managed in the same way. This allows all CI/CD practices to be applied to any change made to an application. As with the source code, all changes are managed via Git, which in turn standardized workflows, easy rollbacks, along with detailed auditing. .

Cloud Build allows you to execute on the idea of “everything as code” in a few ways. First, with Cloud Build, containers are considered first-class citizens. Second, each build step is executed in a docker container as an instance of docker run. A build step specifies an action that you want Cloud Build to perform. Third, Cloud Build provides pre-built images that you can reference in a Cloud Build config file to execute your tasks.

Establish security early in your pipelines

03

It's important to implement security checks and balances as early as possible in the development life cycle. Research from DevOps Research and Assessment (DORA) shows that by finding security risks before you build artifacts or deploy, you can reduce the time and cost spent to address these risks. To detect risks early, you can implement the following security measures in your pipelines:

Build from preapproved code:

Have the InfoSec team build pre-approved, easy-to-consume libraries, packages, toolchains, and processes for developers and IT operations to use in their work.

Scan for vulnerabilities:

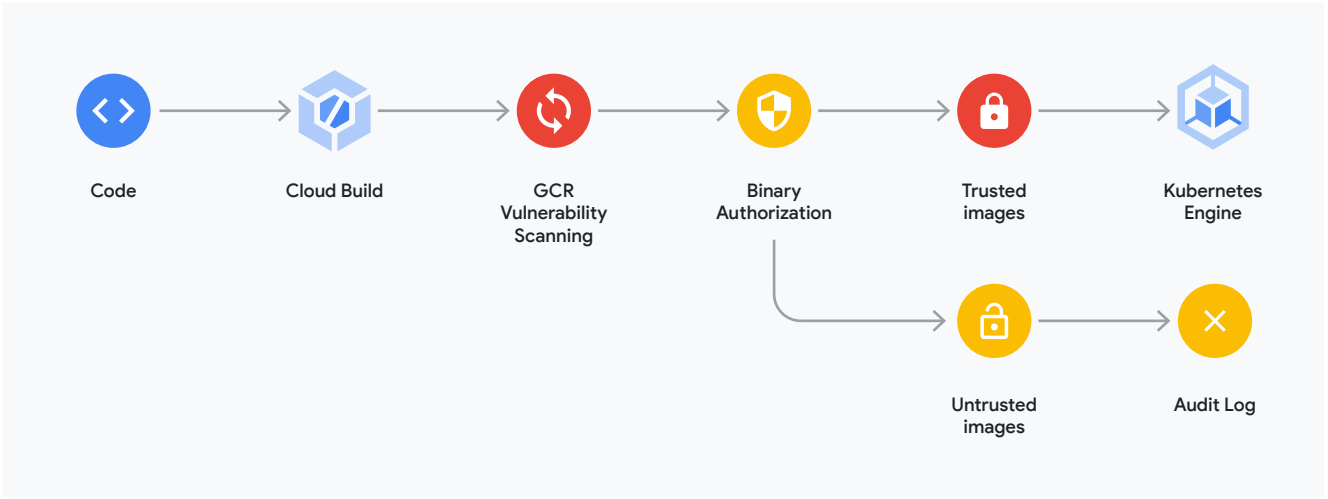
Software vulnerabilities can cause accidental system failures or be intentionally exploited. This is precisely why images created by Cloud Build are automatically scanned for vulnerabilities. It supports package vulnerability scanning for Linux distributions and obtains CVE data from the sources including Debian, Ubuntu, Alpine, the National Vulnerability Database, Red Hat Enterprise Linux, Red Hat and CentOS.

Don't deploy images that contain vulnerabilities:

This entails having a deploy-time security control that ensures only trusted container images are deployed. For instance, by integrating Binary Authorization and Container Registry vulnerability scanning, you can gate deployments based on vulnerability scan findings as part of the overall deploy policy. Binary Authorization also lets you require attestations from different entities or systems. For example, these attestations could include the following:

- Passed vulnerability scan
- Passed QA testing
- Sign off from product owner

Here is qwiklab to get started with integrating vulnerability scanning as a part of your CI/CD pipelines and to ensure images with attestations are deployed to a Kubernetes environment.



Invest in a serverless CI/CD pipeline

04

Serverless solutions have proved themselves to be indispensable in recent times. Organizations of all sizes and industries leverage serverless solutions to introduce brand new products and deliver new features to respond to market demands. In particular, serverless's pay-per-use along with automatic scaling allows teams to meet surges while reducing operating expenses. We've seen more and more CI/CD pipelines delivering into serverless runtimes, primarily because businesses don't want to spend their valuable engineering resources building, maintaining and operating their application delivery infrastructure.

One way to leverage a serverless CI/CD pipeline is to use Cloud Build, which scales up and scales down in response to load, with no need to pre-provision servers or pay in advance for additional capacity. Most importantly, with Cloud Build, you only pay for what you use.

Measure the right metrics for improvement

05

What you can't measure, you can't improve. Hence, it is critical to have the right metrics to measure the effectiveness of software development and delivery practices. DORA has identified and validated a set of capabilities that drive higher software delivery and organizational performance. These metrics include:

01

Deployment Frequency

How often an organization successfully releases to production

02

Lead Time for Changes

The amount of time it takes a commit to get into production

03

Change Failure Rate

The percentage of deployments causing a failure in production

04

Deployment Frequency

How often an organization successfully releases to production

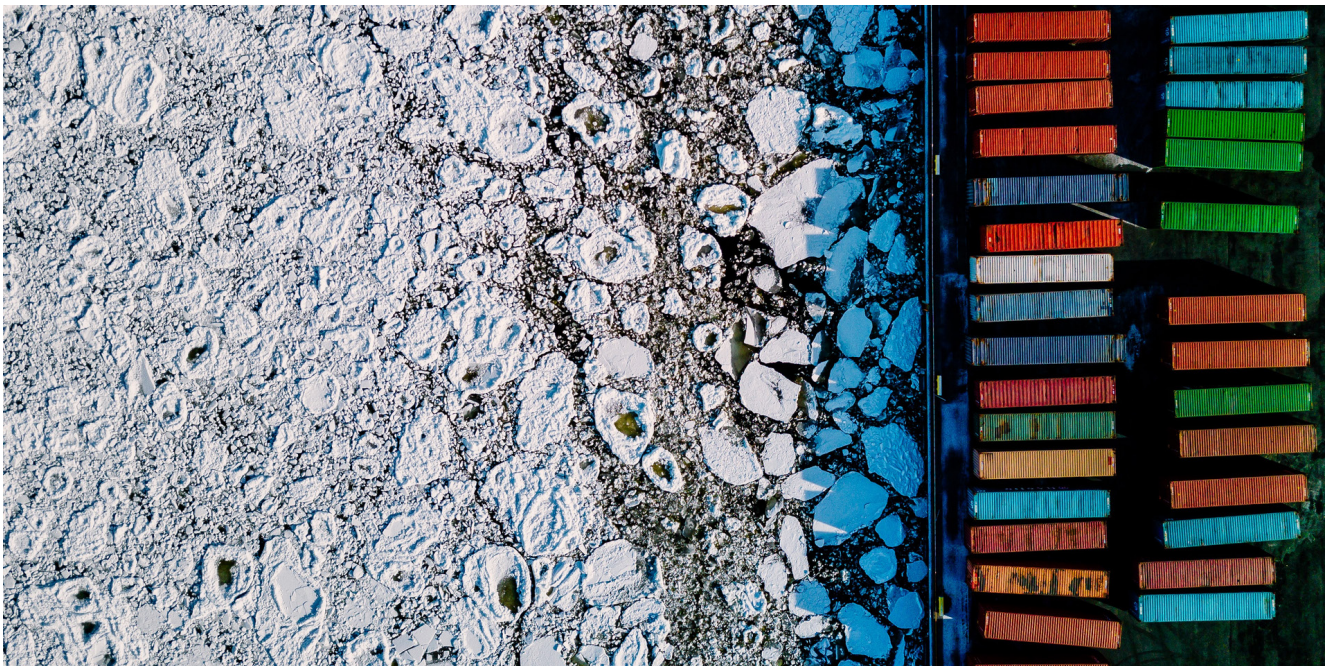
With the Four Keys open source project, you can measure these metrics, and continuously iterate to improve on them. That way, your team can achieve significantly better business outcomes. The Four Keys project automatically sets up a data ingestion pipeline from your GitHub or GitLab repos through Google Cloud services and into Google DataStudio. It then aggregates your data and compiles it into a dashboard with these key metrics, which you can use to track your progress over time.

RUN

Google Kubernetes Engine (GKE)

Google Kubernetes Engine is a managed environment for deploying containerized applications, bringing our latest innovations in developer productivity, resource efficiency, automated operations, and open source flexibility to accelerate your time to market and lower your infrastructure costs. With GKE, you can easily manage and solve the needs of speed, scale, security, and availability of your business.

This section will outline some of the key considerations when running applications in GKE, along with the tools and capabilities that can help you run your Kubernetes workloads securely, cost effectively, and ultimately, more easily.



Running easily

Running and managing a Kubernetes cluster can be challenging. While working with a cluster “from scratch” can be a great learning exercise or a good solution for some highly specialized workloads, often the details of cluster management can be made easier by utilizing a managed service offering. Google Kubernetes Engine makes running Kubernetes easier with convenient tooling and integrations along with the convenience of offering Kubernetes clusters as a managed service.

In addition, with the launch of GKE Autopilot, GKE users can choose from two different modes of operation, each of which provides its own level of control. Autopilot’s hands-off fully managed Kubernetes experience allows you to focus more on your workloads and less on managing cluster infrastructure. Watch a demo of Autopilot in action here.

Running securely

GKE provides and works with a variety of tools to control accessibility across your Kubernetes infrastructure, allowing for a secure, zero-trust environment, in which each user only has the exact level of access they need — nothing more and nothing less. With GKE, you can:

- Grant users fine-grained access to your cluster with namespaces and Kubernetes RBAC
- Isolate pods with GKE sandbox
- Restrict pod-to-pod access with a network policies or Anthos Service Mesh
- Restrict pod-to-Google access with Workload Identity
- Use policies to prevent deviations from approved security baselines

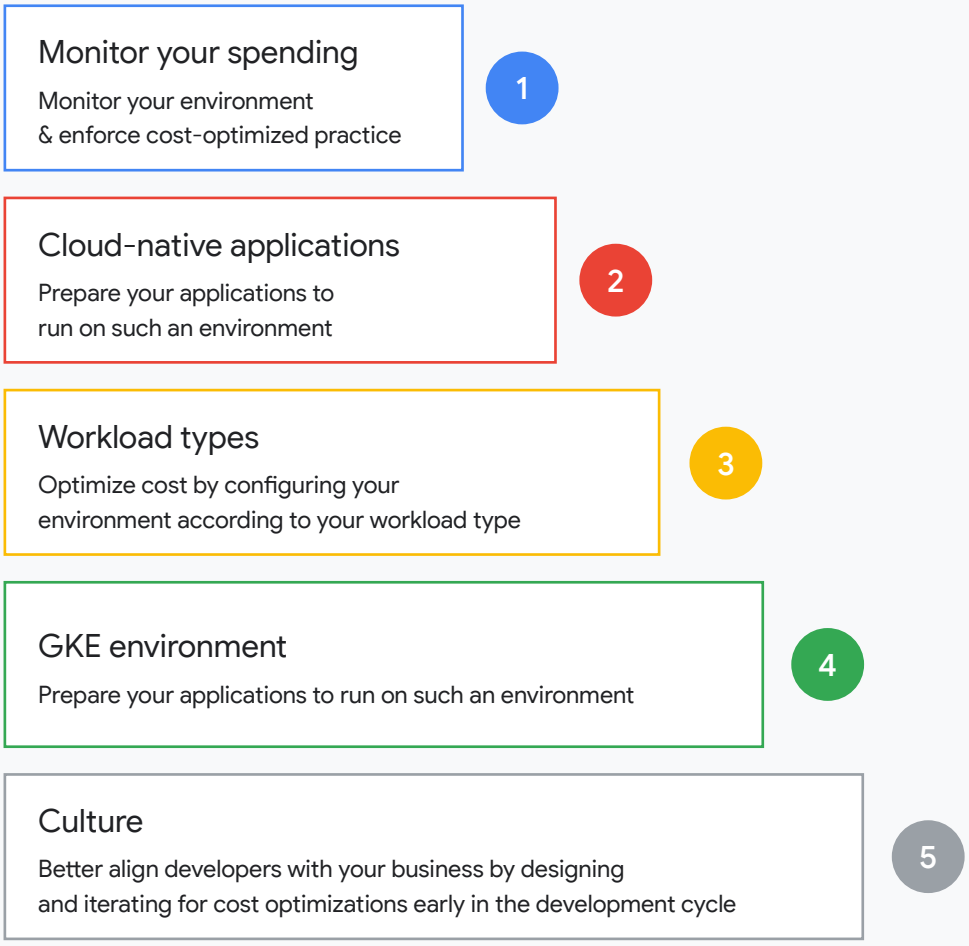
For more information about how to harden your GKE cluster, see https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster.

Running cost-effectively

Achieving a balance between cost and performance for your workloads (and subsequently your business) is done by promoting a culture of cost-awareness across your teams. While it's tempting to only consider cost at the beginning of your cloud journey, it is critical to understand the capabilities of the environment that your applications are running in - in this context, the GKE environment.

Finding that balance between cost, performance and reliability in GKE means tuning features and configurations such as autoscaling, machine types, regions, etc. The right configuration depends on the nature of your workloads, and you'll achieve the best outcome if you understand your workloads' characteristics, (e.g., user-facing applications, batch jobs, etc.) and match them to the environment's capabilities. Finally, you must monitor your spending and create guardrails so that you can enforce best practices early in your development cycle.

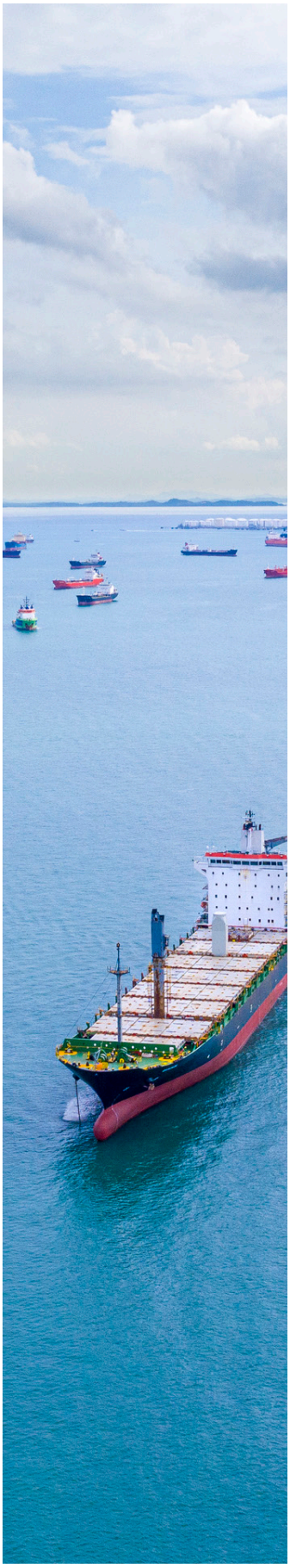
Here is a good way to visualize these considerations:



Starting with a cultural foundation that emphasizes cost optimization early in the development lifecycle, GKE brings further cost efficiencies with features like auto-scaling and rightsizing and allows you to configure your environment according to your workload type.

For more information about how to build cost-optimized applications, read more about our best practices and check here for the most up-to-date content. You can also watch our cost optimization video series or just jump right in with this Qwiklabs quest!

Running with scale



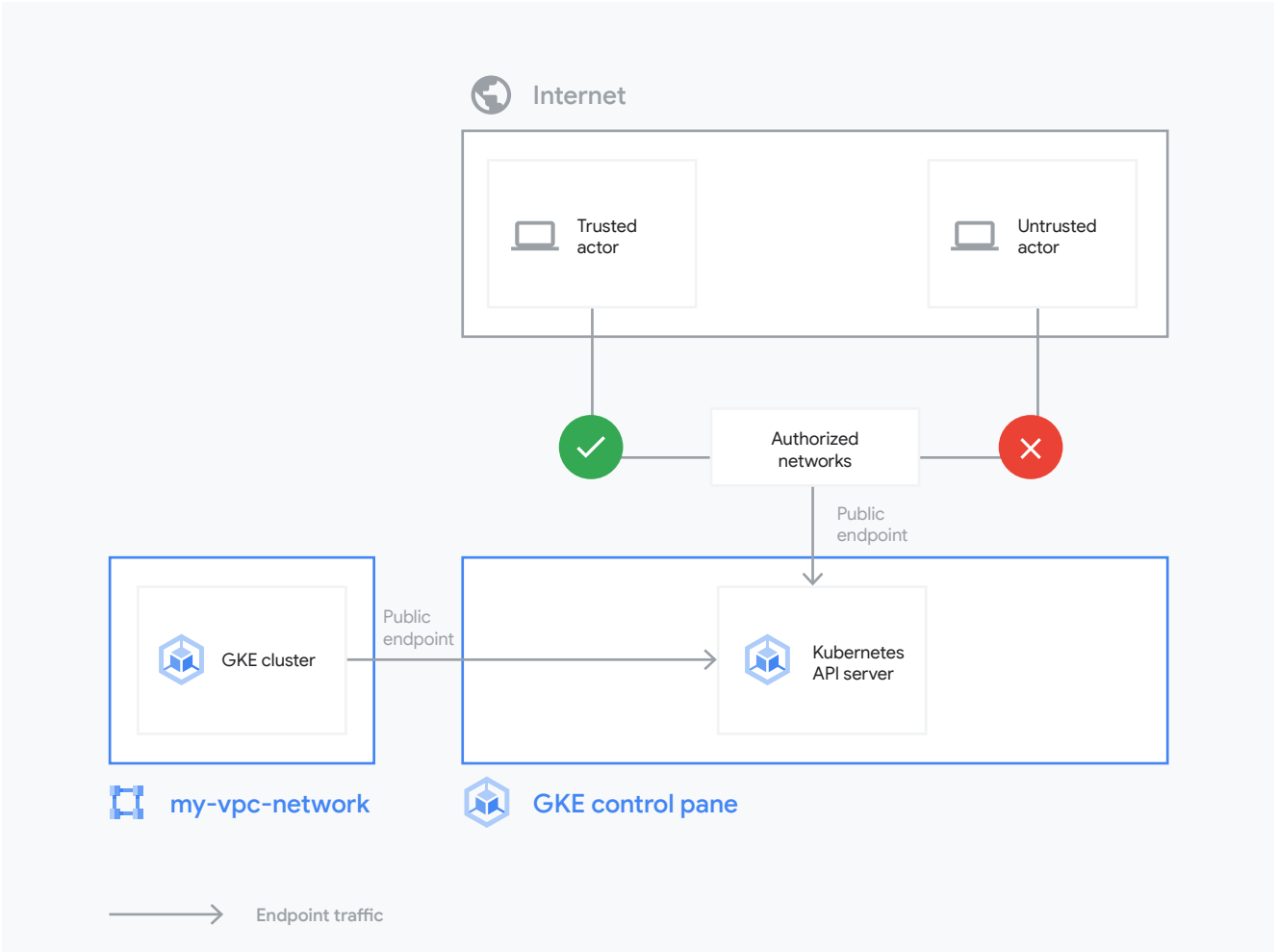
In a Kubernetes cluster, scalability refers to the ability of the cluster to grow while staying within its service-level objectives (SLOs). Kubernetes is a complex system, and its ability to scale is determined by multiple factors. Some of these factors include the type and number of nodes in a node pool, the types and numbers of node pools, the number of Pods available, how resources are allocated to Pods, and the number of Services or backends behind a Service. GKE is the most scalable Kubernetes service, supporting 15k node clusters. Here are some important settings that will help you set started with running with scale:

- Use regional clusters for all production workloads
- Always use a VPC-native network
- Use private clusters to avoid using large numbers of unnecessary external IP addresses
- Use NodeLocal DNS cache to avoid DNS reliability problems
- Upgrade frequently and try running on the highest version as possible (while still balancing your upgrades with the need for longer soak time of releases)

Read about more best practices for scalability here and watch some videos that cover the basics and go deeper.

Managing the cluster network

Kubernetes allows you to declaratively define how your applications are deployed, how applications communicate with each other and with the Kubernetes control plane, and how clients can reach your applications. Here are some best practices to keep in mind when getting started with GKE networking:



- Use Shared VPC networks and VPC-native clusters. Changes to the VPC design cannot be done after you create a cluster.
- Manage your IP space. GKE clusters are IP address-intensive and require strong awareness of subnet allocation and connectivity options. You usually cannot expand your IP address assignments after you create a cluster.
- Secure your GKE clusters. A few security recommendations include using private clusters, VPC firewall rules, and Network Policy. Use these Network Policy templates to quickly get up to speed. This diagram shows trusted users being able to communicate with the GKE control plane through the public endpoint as they are part of authorized networks, while access from untrusted actors is blocked. Communication to and from the GKE cluster happens through the private endpoint of the control plane.
- Plan for scale and exposure of services. NodeLocal DNSCache, CloudNAT, and container-native load-balancing are a few recommendations for enabling scalable services.
- Ensure proper IAM roles and Cloud Operations are in place to properly control and observe your GKE clusters.
- Observe how your network traffic behaves inside your cluster with Network Policy logging and GKE annotations for VPC Flow Logs

Learn more of the best practices for GKE networking, in this video demo and full reference guide, and dive deeper into exposing GKE applications with Ingress and Services controllers here.

OPERATE

Monitor and troubleshoot

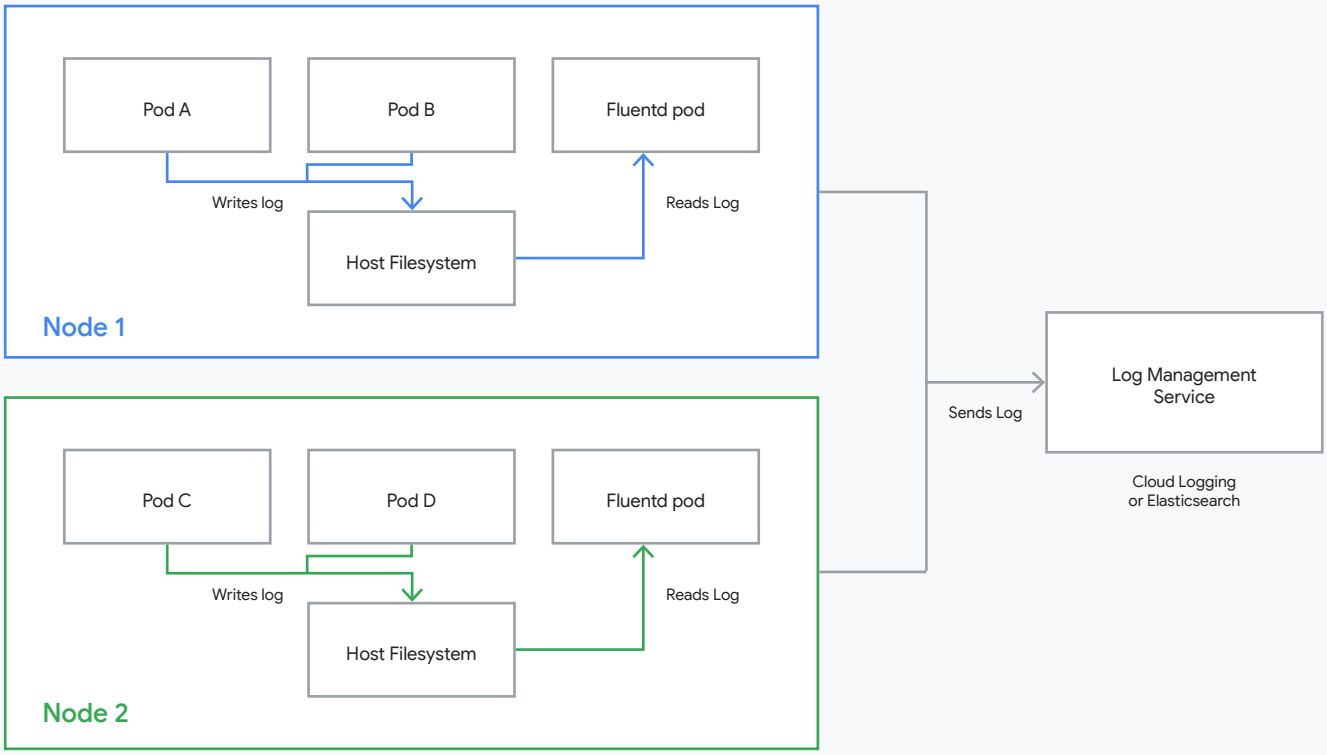
Debugging and troubleshooting are necessary parts of running business-critical applications, however they take your time away from writing code. The best practices outlined below provide a balanced approach to collecting and using data about the health of your systems and applications, while preserving the time and budget of developers and operations.

Build or utilize a log management system

When you run into an issue with your applications, logs provide data about point-in-time events, enabling you to troubleshoot, perform a root cause analysis, and remediate. This is especially important as your environments scale to hundreds or thousands of containers.

Cloud platforms have tools that will automatically capture system logs such as audit logs and service logs, however with unmanaged Kubernetes, you need to build a log management system (see diagram below) so your application logs can be captured and exported to an analysis/visualization tool.

Diagram of a typical log management system in Kubernetes (more details)



Managed Kubernetes services such as GKE offer a log management system, by default, with no configuration necessary. When you deploy a managed Kubernetes cluster, a per-node logging agent is installed which reads container logs, adds metadata, and then stores them in a persistent datastore. Resources in a Kubernetes cluster are often wiped and rebuilt so this export to a persistent datastore is necessary so logs are not lost.

Applications running on containers need to be configured so their logs are captured by a logging agent. The GKE logging agent checks for logs in the Standard output (stdout) and standard error logs (stderr) from containerized processes, kubelet, container runtime logs, and logs for system components. To make sure your container application logs are captured, use the native logging mechanisms of containers and write structured logs as JSON objects serialized on a single line to the aforementioned stdout or stderr (read more about structured versus unstructured logs). Alternatively, you can use an idiomatic logging library to capture application logs, as many language runtimes are integrated into GKE.

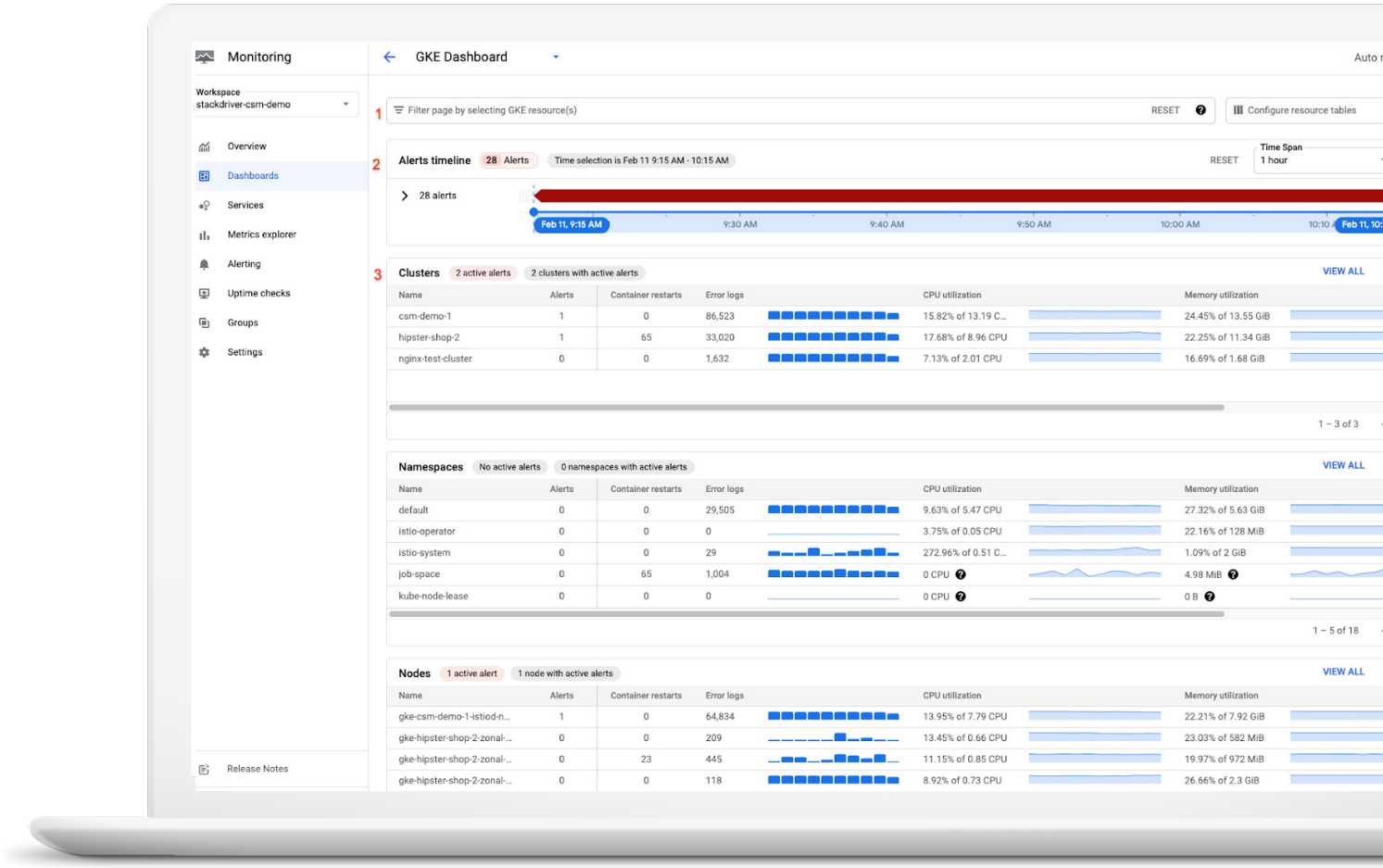
When you're ready to analyze your logs, you can use the kubectl command to get a quick view, but you need a log management service for more sophisticated queries, filtering options and time selections. If you are using a managed Kubernetes service like GKE, you can use the built-in logging tool of your service to analyze your logs.

Be proactive with effective container monitoring

Setting yourself and your customers up for success means monitoring your applications and infrastructure. Monitoring helps you quickly recognize when an issue is occurring so you can find and troubleshoot the affected resource.

Kubernetes clusters have many components and it is essential to have visibility into each of the components. Therefore you should configure dashboards, or use one out of the box from a managed Kubernetes service, which gives you the ability to view your fleet by each Kubernetes component: cluster, namespace, node, workloads, services, pods, and containers. In order to investigate issues that occur over time, you should be able to filter on resources as well as a time span. For each resource you should see the display name of the resource, the number of open and acknowledged alerts (see below) for that resource and its children, container restarts, Error logs counts, CPU utilization, memory utilization, and the disk utilization of pods that can be attributed to a resource.

When you have this granular view of your metrics, create alerts so you get a proactive notification when the metrics fall outside of acceptable ranges. Monitoring tools that are native to a cloud service can feature deep integrations that allow you to navigate from alerts to a monitoring dashboard, then from that dashboard to resource logs, allowing for faster resolution (see the GKE Dashboard from Cloud Monitoring below as an example).

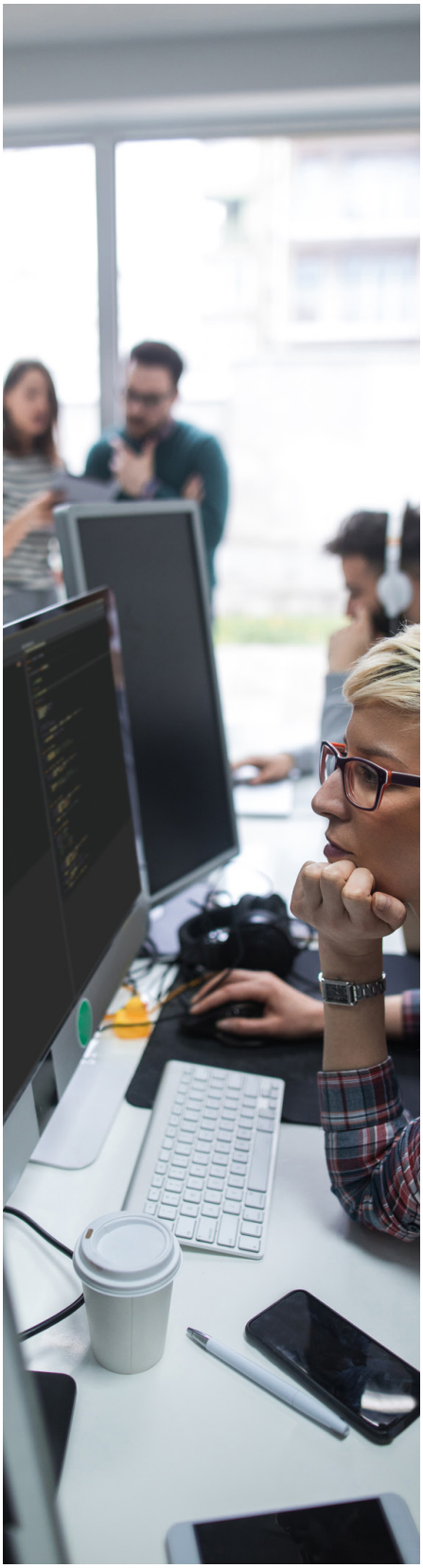


A view of the GKE Dashboard in Cloud Operations. Sections: 1) the filter bar to select which GKE resources to filter on within the dashboard, 2) integrated alerts timeline, 3) detailed breakdown of clusters, namespaces, nodes, workloads, services, pods, and containers.

Take monitoring to the next level

Proactive monitoring and alerting are useful for flagging issues, but they may not help drive a shared sense of responsibility between developers and operators, where developers seek greater speed in releasing new features and operations teams strive for greater stability. Service Level Objectives (SLOs) are agreed-upon levels of application or service uptime that teams are mutually accountable to achieve. In addition to contributing to customer success, SLOs take the burden off of solving alerts that are not leading to SLO violations.

Instrument your applications for deeper observability



Single operations within an application are termed spans, and the collection of operations that make up a transaction is called a trace. You can monitor and troubleshoot traces for latency when your applications are instrumented to capture and export them.

Use an open-source telemetry tool such as OpenTelemetry or its predecessor OpenCensus to instrument your applications for trace observability. Many open-source projects and vendors, such as Google Cloud's operations suite, have built integrations to ingest the data so you are not locked into one vendor's monitoring tools.

Improving your application development velocity and preparing for scale are likely some of the reasons why you're choosing to build on containers. These best practices lay the foundation for development or DevOps teams to act quickly and efficiently in fast-paced, large-scale environments. Read more about Google Cloud's operations suite for GKE.

Summary

As a developer on Google Cloud, you can take advantage of the streamlined end-to-end experience:

1

Speed up Kubernetes development with tools such as Cloud Code that will make it easier for you to write code and be more productive.

2

Accelerate Kubernetes deployment with end-to-end automation from source to production with Google Cloud CI/CD tools.

3

Run Kubernetes easily with GKE Autopilot or Standard mode of operations, depending on the level of flexibility, responsibility, and control you want over your Kubernetes cluster.

4

Scale observability of Kubernetes workloads by utilizing Google Cloud's operations suite.

For further reading, check out Re-architecting to cloud native: an evolutionary approach to increasing developer productivity at scale.