Solutions (https://cloud.google.com/solutions/) Solutions

Serving websites

This article discusses how to host a website on Google Cloud. Google Cloud provides a robust, flexible, reliable, and scalable platform for serving websites. Google built Google Cloud by using the same infrastructure that Google uses to serve content from sites such as Google.com, YouTube, and Gmail. You can serve your website's content by using the type and design of infrastructure that best suits your needs.

You might find this article useful if you are:

- Knowledgeable about how to create a website and have deployed and run some web-serving infrastructure before.
- Evaluating whether and how to migrate your site to Google Cloud.

If you want to build a simple website, consider using Google Sites (https://chrome.google.com/webstore/detail/google-sites/gmandedkgonhldbnjpikffdnneenijnd?), a structured wiki- and web page-creation tool. For more information, visit Sites help (https://support.google.com/sites/?hl=en#topic=7184580).

Note: You might find it helpful to read the Concepts page of the Google Cloud overview (https://cloud.google.com/docs/overview) before reading this article. Some of those concepts are referenced in this article without further explanation. If you're already a bit familiar with Google Cloud, you can skip this step.

Choosing an option

If you're new to using Google Cloud, it's a reasonable approach to start by using the kind of technology you're already familiar with. For example, if you currently use hardware servers or virtual machines (VMs) to host your site, perhaps with another cloud provider or on your own hardware, Compute Engine (#compute-engine) provides a familiar paradigm for you. If you already use a platform-as-a-service (PaaS) offering, such as Heroku or Engine Yard, App Engine (#app-engine) might be the best place to start.

After you become more familiar with Google Cloud, you can explore the richness of products and services that Google Cloud provides. For example, if you started by using Compute Engine, you might augment your site's capabilities by using Google Kubernetes Engine (GKE) (#kubernetes-engine) or migrate some or all of the functionality to App Engine.

The following table summarizes your hosting options on Google Cloud:

Option	Product	Data storage	Load balancing	Scalability	Logging
Static website (#static-site)	Cloud Storage Firebase Hosting	Cloud Storage bucket	n/a	Automatically	n/a

Option	Product	Data storage	Load balancing	Scalability	Logging
Virtual machines (#compute-engine)	Compute Engine	Cloud SQL Admin API, Cloud Storage API, Datastore API, and Cloud Bigtable API, or you can use another external storage provider. Hard-disk-based persistent disks, called *standard persistent disks*, and solid-state persistent disks (SSD).	HTTP(S) TCP Proxy SSL Proxy IPv6 termination Network Cross-region Internal	Automatically with managed instance groups	Stackdriver Logging (https://cloud.google.com/logging/docs/) Stackdriver Monitoring (https://cloud.google.com/monitoring/docs/) Monitoring Console (https://app.google.stackdriver.com/)
Containers (#kubernetes-engine)	GKE	Similar to Compute Engine but interacts with persistent disks differently	Network HTTP(S)	Cluster autoscaler	Stackdriver Logging (https://cloud.google.com/logging/docs/) Stackdriver Monitoring (https://cloud.google.com/monitoring/docs/) Monitoring Console (https://app.google.stackdriver.com/)
Managed platform (#app-engine)	App Engine	Google does it for you	Google does it for you	Google does it for you	Google does it for you

This article can help you to understand the main technologies that you can use for web serving on Google Cloud and give you a glimpse of how the technologies work. The article provides links to complete documentation, tutorials, and solutions articles that can help you build a deeper understanding, when you're ready.

Understanding costs

Because there are so many variables and each implementation is different, it's beyond the scope of this article to provide specific advice about costs. To understand Google's principles about how pricing works on Google Cloud, see the pricing page (https://cloud.google.com/pricing/principles). To understand pricing for individual services, see the product pricing section (https://cloud.google.com/pricing/list). You can also use the pricing calculator (https://cloud.google.com/products/calculator/) to estimate what your Google Cloud usage might look like. You can provide details about the services you want to use and then see a pricing estimate.

Setting up domain name services

Usually, you will want to register a domain name for your site. You can use a public domain name registrar, such as Google Domains (https://domains.google.com/about/), to register a unique name for your site. If you want complete control of your own domain name system (https://wikipedia.org/wiki/Domain_Name_System) (DNS), you can use Cloud DNS (https://cloud.google.com/dns) to serve as your DNS provider. The Cloud DNS documentation includes a quickstart (https://cloud.google.com/dns/quickstart) to get you going.

If you have an existing DNS provider that you want to use, you generally need to create a couple of records with that provider. For a domain name such as `example.com`, you create an `A` record with your DNS provider. For the `www.example.com` sub-domain, you create a `CNAME` record for `www` to point it to the `example.com` domain. The `A` record maps a hostname to an IP address. The `CNAME` record creates an alias for the `A` record.

If your domain name registrar is also your DNS provider, that's probably all you need to do. If you use separate providers for registration and DNS, make sure that your domain name registrar has the correct name servers associated with your domain.

After making your DNS changes, the record updates can take some time to propagate depending on your time-to-live (TTL) values in your zone. If this is a new hostname, the changes go into effect quickly because the DNS resolvers don't have cached previous values and can contact the DNS provider to get the necessary information to route requests.

Hosting a static website

The simplest way to serve website content over HTTP(S) is to host *static web pages*. Static web pages are served unchanged, as they were written, usually by using HTML. Using a static website is a good option if your site's pages rarely change after they have been published, such as blog posts or pages that are part of a small-business website. You can do a lot with static web pages, but if you need your site to have robust interactions with users through server-side code, you should consider the other options discussed in this article.

Hosting a static website with Cloud Storage

Note: Though Cloud Storage serves content over HTTPS, it doesn't support end-to-end HTTPS for custom domains. If you need end-to-end HTTPS serving, check out Firebase hosting (#firebase_hosting) in the next section.

To host a static site in Cloud Storage, you need to create a Cloud Storage bucket (https://cloud.google.com/storage/docs/key-terms#buckets), upload the content, and test your new site. You can serve your data directly from `storage.googleapis.com` (https://cloud.google.com/storage/docs/cloud-console#_sharingdata), or you can verify that you own your domain (https://cloud.google.com/storage/docs/domain-name-verification) and use your domain name.

You can create your static web pages however you choose. For example, you could hand-author pages by using HTML and CSS. You can use a *static-site generator*, such as Jekyll (https://jekyllrb.com/), Ghost (https://ghost.org/), or Hugo (https://gohugo.io/), to create the content. With static-site generators, you create a static website by authoring in markdown (https://wikipedia.org/wiki/Markdown), and providing templates and tools. Site generators generally provide a local web server that you can use to preview your content.

After your static site is working, you can update the static pages by using any process you like. That process can be as straightforward as hand-copying an updated page to the bucket. You might choose to use a more automated approach, such as storing your content on GitHub and then using a webhook (https://developer.github.com/webhooks/) to run a script that updates the bucket. An even more advanced system might use a continuous-integration/continuous-delivery (CI/CD) tool, such as Jenkins (https://jenkins.io/), to update the content in the bucket. Jenkins has a Cloud Storage plugin (https://wiki.jenkins-ci.org/display/JENKINS/Google+Cloud+Storage+Plugin) that provides a `Google Cloud Storage Uploader` post-build step to publish build artifacts to Cloud Storage.

If you have a web app that needs to serve static content or user-uploaded static media, using Cloud Storage can be a cost-effective and efficient way to host and serve this content, while reducing the amount of dynamic requests to your web app.

Additionally, Cloud Storage can directly accept user-submitted content. This feature lets users upload large media files directly and in a secure manner (https://cloud.google.com/solutions/architecture/digitalassets), without proxying through your servers.

To get the best performance from your static website, see Best practices for Cloud Storage (https://cloud.google.com/storage/docs/best-practices).

For more information, see the following pages:

- Hosting a static website (https://cloud.google.com/storage/docs/website-configuration)
- Jekyll Static Website on Cloud Storage (http://little418.com/2015/07/jekyll-google-cloud-storage.html) (blog post)
- J is for Jenkins (https://medium.com/google-cloud/a-to-z-of-google-cloud-platform-a-personal-selection-j-is-for-jenkins-7a718d1f458) (blog post)
- Band Aid 30 on Google Cloud (https://cloudplatform.googleblog.com/2015/04/BandAid-30-on-Google-Cloud-Platform-Theres-more-than-one-way-to-skin-a-Web-server.html) (blog post)
- Cloud Storage documentation (https://cloud.google.com/storage/docs/overview)

Hosting a static website with Firebase Hosting

Firebase Hosting provides fast and secure static hosting for your web app. With Firebase Hosting, you can deploy web apps and static content to a global content-delivery network (CDN) by using a single command.

Here are some benefits you get when you use Firebase Hosting:

- Zero-configuration SSL is built into Firebase Hosting. Provisions SSL certificates on custom domains for free.
- All of your content is served over HTTPS.
- Your content is delivered to your users from CDN edges around the world.
- Using the Firebase CLI (https://firebase.google.com/docs/cli/), you can get your app up and running in seconds. Use command-line tools to add deployment targets into your build process.
- You get release management features, such as atomic deployment of new assets, full versioning, and one-click rollbacks.
- Hosting offers a configuration useful for single-page apps (https://firebase.google.com/docs/hosting/url-redirects-rewrites) and other sites that are more app-like.
- Hosting is built to be used seamlessly with other Firebase features.

For more information, see the following pages:

- Firebase Hosting guide (https://firebase.google.com/docs/hosting)
- Get started with Firebase Hosting (https://firebase.google.com/docs/hosting/quickstart)

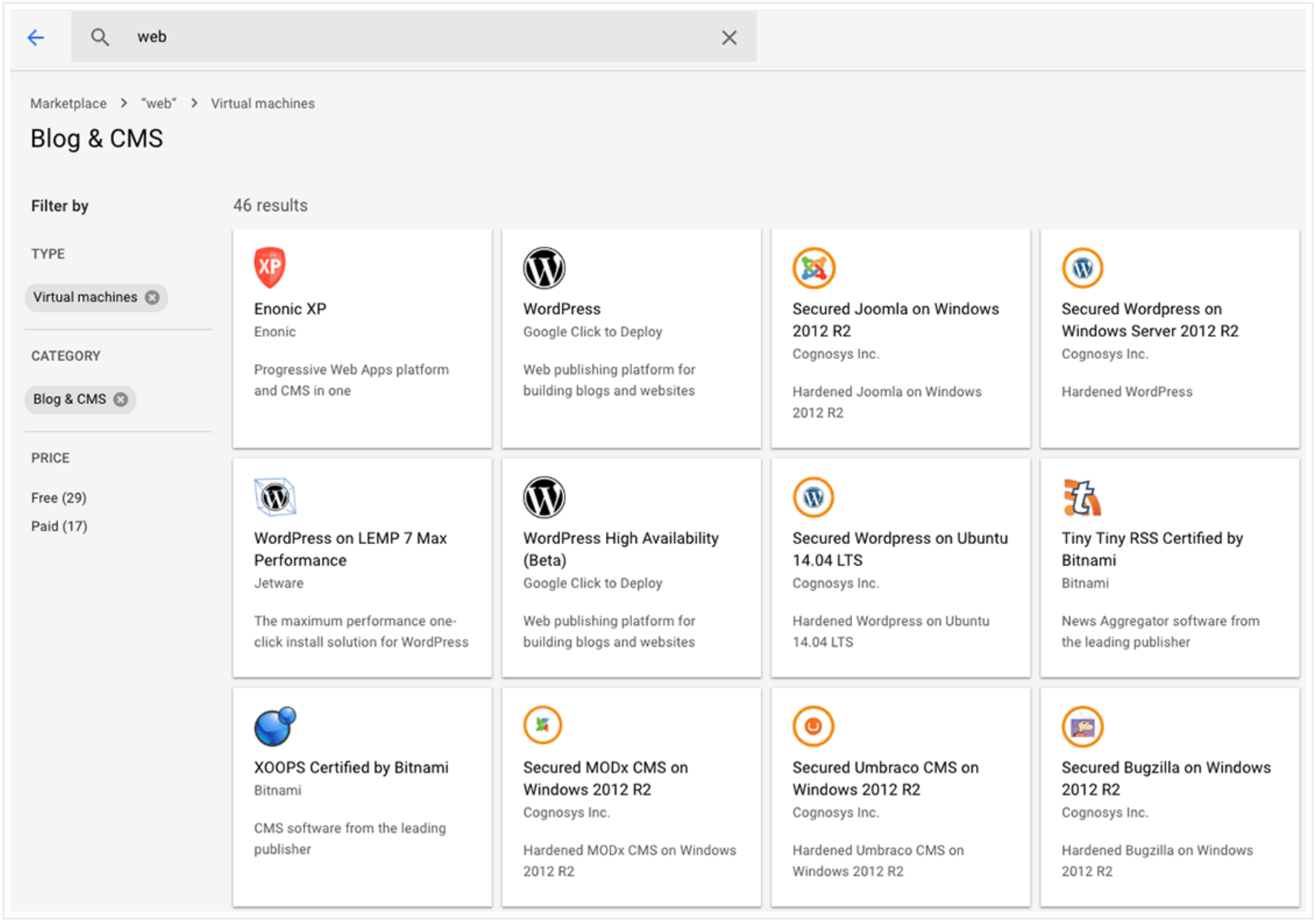
Using virtual machines with Compute Engine

For infrastructure as a service (IaaS) use cases, Google Cloud provides Compute Engine (https://cloud.google.com/compute/). Compute Engine provides a robust computing infrastructure, but you must choose and configure the platform components that you want to use. With Compute Engine, it's your responsibility to configure, administer, and monitor the systems. Google ensures that resources are available, reliable, and ready for you to use, but it's up to you to provision and manage them. The advantage, here, is that you have complete control of the systems and unlimited flexibility.

Use Compute Engine to design and deploy nearly any website-serving system you want. You can use VMs, called instances (https://cloud.google.com/compute/docs/instances), to build your app, much like you would if you had your own hardware infrastructure. Compute Engine offers a variety of machine types (https://cloud.google.com/compute/docs/machine-types) to customize your configuration to meet your needs and your budget. You can choose which operating systems, development stacks, languages, frameworks, services, and other software technologies you prefer.

Setting up automatically with Google Cloud Marketplace

The easiest way to deploy a complete web-serving stack is by using Google Cloud Marketplace (https://cloud.google.com/marketplace/). With just a few clicks, you can deploy any of over 100 fully realized solutions with Google Click to Deploy or Bitnami.



Blog & CMS

Filter by

TYPE

Virtual machines

CATEGORY

Blog & CMS

PRICE

Free (29)

Paid (17)

46 results

Enonic XP
Enonic
Progressive Web Apps platform and CMS in one

WordPress
Google Click to Deploy
Web publishing platform for building blogs and websites

Secured Joomla on Windows 2012 R2
Cognosys Inc.
Hardened Joomla on Windows 2012 R2

Secured Wordpress on Windows Server 2012 R2
Cognosys Inc.
Hardened WordPress

WordPress on LEMP 7 Max Performance
Jetware
The maximum performance one-click install solution for WordPress

WordPress High Availability (Beta)
Google Click to Deploy
Web publishing platform for building blogs and websites

Secured Wordpress on Ubuntu 14.04 LTS
Cognosys Inc.
Hardened Wordpress on Ubuntu 14.04 LTS

Tiny Tiny RSS Certified by Bitnami
Bitnami
News Aggregator software from the leading publisher

XOOPS Certified by Bitnami
Bitnami
CMS software from the leading publisher

Secured MODx CMS on Windows 2012 R2
Cognosys Inc.
Hardened MODx CMS on Windows 2012 R2

Secured Umbraco CMS on Windows 2012 R2
Cognosys Inc.
Hardened Umbraco CMS on Windows 2012 R2

Secured Bugzilla on Windows 2012 R2
Cognosys Inc.
Hardened Bugzilla on Windows 2012 R2

For example, you can set up a LAMP stack (https://cloud.google.com/marketplace/solution/click-to-deploy-images/lamp) or WordPress (https://cloud.google.com/marketplace/solution/click-to-deploy-images/wordpress) with Google Cloud

Marketplace. The system deploys a complete, working software stack in just a few minutes on a single instance. Before you deploy, Google Cloud Marketplace shows you cost estimates for running the site, gives you clear information about which versions of the software components it installs for you, and lets you customize your configuration by changing component instance names, choosing the machine type, and choosing a disk size. After you deploy, you have complete control over the Compute Engine instances, their configurations, and the software.

Setting up manually

You can also create your infrastructure on Compute Engine manually, either building your configuration from scratch or building on a Google Cloud Marketplace deployment. For example, you might want to use a version of a software component not offered by Google Cloud Marketplace, or perhaps you prefer to install and configure everything on your own.

Providing a complete framework and best practices for setting up a website is beyond the scope of this article. But from a high-level view, the technical side of setting up a web-serving infrastructure on Compute Engine requires that you:

- **Understand the requirements**. If you're building a new website, make sure you understand the components you need, such as instances, storage needs, and networking infrastructure. If you're migrating your app from an existing solution, you probably already understand these requirements, but you need think through how your existing setup maps to Google Cloud services (https://cloud.google.com/docs/overview/cloud-platform-services).
- **Plan the design**. Think through your architecture and write down your design. Be as explicit as you can.
- **Create the components**. The components that you might usually think of as physical assets, such as computers and network switches, are provided through services in Compute Engine. For example, if you want a computer, you have to create a Compute Engine instance. If you want a persistent hard disk drive, you create that, too. Cloud Deployment Manager (https://cloud.google.com/deployment-manager/overview) makes this an easy and repeatable process.
- **Configure and customize.** After you have the components you want, you need to configure them, install and configure software, and write and deploy any customization code that you require. You can replicate the configuration by running shell scripts, which helps to speed future deployments. Deployment Manager helps here, too, by providing declarative, flexible configuration templates for automatic deployment of resources. You

can also take advantage of IT automation tools such as Puppet (https://puppet.com/) and Chef (https://www.chef.io/).

- **Deploy the assets**. Presumably, you have web pages and images.
- **Test**. Verify that everything works as you expect.
- **Deploy to production**. Open up your site for the world to see and use.

To help you to get started and understand what it's like to set up Compute Engine instances manually, try one or more of the following tutorials:

- Hosting a website using LAMP (https://cloud.google.com/compute/docs/tutorials/setup-lamp)
- Setting up Joomla! (https://cloud.google.com/compute/docs/tutorials/setup-joomla)
- Setting up Drupal (https://cloud.google.com/solutions/drupal/setup-drupal)

Storing data with Compute Engine

Most websites need some kind of storage. You might need storage for a variety of reasons, such as saving files that your users upload, and of course the assets that your site uses.

Google Cloud provides a variety of managed storage services, including:

- A SQL database in Cloud SQL (https://cloud.google.com/sql/docs/introduction), which is based on MySQL.
- Two options for NoSQL data storage: Datastore (https://cloud.google.com/datastore/docs/concepts/overview) and Cloud Bigtable (https://cloud.google.com/bigtable/docs/).
- Consistent, scalable, large-capacity object storage in Cloud Storage (https://cloud.google.com/storage/docs/overview). Cloud Storage comes in several classes:
  - Standard provides maximum availability.
  - Nearline provides a low-cost choice ideal for data accessed less than once a month.
  - Coldline provides a low-cost choice ideal for data accessed less than once a quarter.
  - Archive provides the lowest-cost choice for archiving, backup, and disaster recovery.
- Persistent disks on Compute Engine (https://cloud.google.com/compute/docs/disks/#pdspecs) for use as primary storage for your instances. Compute Engine offers both hard-disk-based persistent disks, called *standard persistent disks*, and solid-state persistent disks (SSD). You can also choose to set up your preferred storage technology on Compute

Engine by using persistent disks. For example, you can set up PostgreSQL (https://cloud.google.com/solutions/setup-postgres) as your SQL database or MongoDB (https://cloud.google.com/solutions/mongodb/intro) as your NoSQL storage. To understand the full range and benefits of storage services on Google Cloud, see Choosing a storage option (https://cloud.google.com/docs/storing-your-data).

Load balancing with Compute Engine

For any website that operates at scale, using load-balancing technologies to distribute the workload among servers is often a requirement. You have a variety of options when architecting your load-balanced web servers on Compute Engine, including:

- HTTP(S) load balancing (https://cloud.google.com/compute/docs/load-balancing/http/). Explains the fundamentals of using Cloud Load Balancing.
  - Content-based load balancing (https://cloud.google.com/compute/docs/load-balancing/http/content-based-example). Demonstrates how to distribute traffic to different instances based on the incoming URL.
  - Cross-region load balancing (https://cloud.google.com/compute/docs/load-balancing/http/cross-region-example). Demonstrates configuring VM instances in different regions and using HTTP or HTTPS load balancing to distribute traffic across the regions.
- TCP Proxy load balancing (https://cloud.google.com/load-balancing/docs/tcp/). Demonstrates setting up global TCP Proxy load balancing for a service that exists in multiple regions.
- SSL Proxy load balancing (https://cloud.google.com/compute/docs/load-balancing/tcp-ssl). Demonstrates setting up global SSL Proxy load balancing for a service that exists in multiple regions.
- IPv6 termination for HTTP(S), SSL Proxy, and TCP Proxy load balancing (https://cloud.google.com/compute/docs/load-balancing/ipv6). Explains IPv6 termination and the options for configuring load balancers to handle IPv6 requests.
- Network load balancing (https://cloud.google.com/compute/docs/load-balancing/network/example). Shows a basic scenario that sets up a layer 3 load balancing configuration to distribute HTTP traffic across healthy instances.

- Cross-region load balancing using Microsoft IIS backends (https://cloud.google.com/compute/docs/tutorials/http-load-balancing-iis). Shows how to use the Compute Engine load balancer to distribute traffic to Microsoft Internet Information Services (IIS) servers.
- Setting up internal load balancing (https://cloud.google.com/compute/docs/load-balancing/internal/) You can set up a load balancer that distributes network traffic on a private network that isn't exposed to the internet. Internal load balancing is useful not only for intranet apps where all traffic remains on a private network, but also for complex web apps where a frontend sends requests to backend servers by using a private network.

Load balancing deployment is flexible, and you can use Compute Engine with your existing solutions. For a few examples, see Autoscaled internal load balancing using HAProxy and Consul (https://cloud.google.com/solutions/autoscaled-load-balancing-using-haproxy-and-consul-on-compute-engine) for information about autoscaling both the HAProxy load balancing tier and the backend server tier. See HTTP(S) load balancing using NGINX (https://cloud.google.com/solutions/https-load-balancing-nginx) for one possible solution that you could use in place of the Compute Engine load balancer.

Content distribution with Compute Engine

Because response time is a fundamental metric for any website, using a CDN to lower latency and increase performance is often a requirement, especially for a site with global web traffic.

Cloud CDN uses Google's globally distributed edge points of presence to deliver content from cache locations closest to users. Cloud CDN works with HTTP(S) load balancing. To serve content out of Compute Engine, Cloud Storage, or both from a single IP address, enable Cloud CDN (https://cloud.google.com/cdn/docs/using-cdn) for an HTTP(S) load balancer.

Autoscaling with Compute Engine

You can set up your architecture to add and remove servers as demand varies. This approach can help to ensure that your site performs well under peak load, while keeping costs under control during more-typical demand periods. Compute Engine provides an autoscaler that you can use for this purpose.

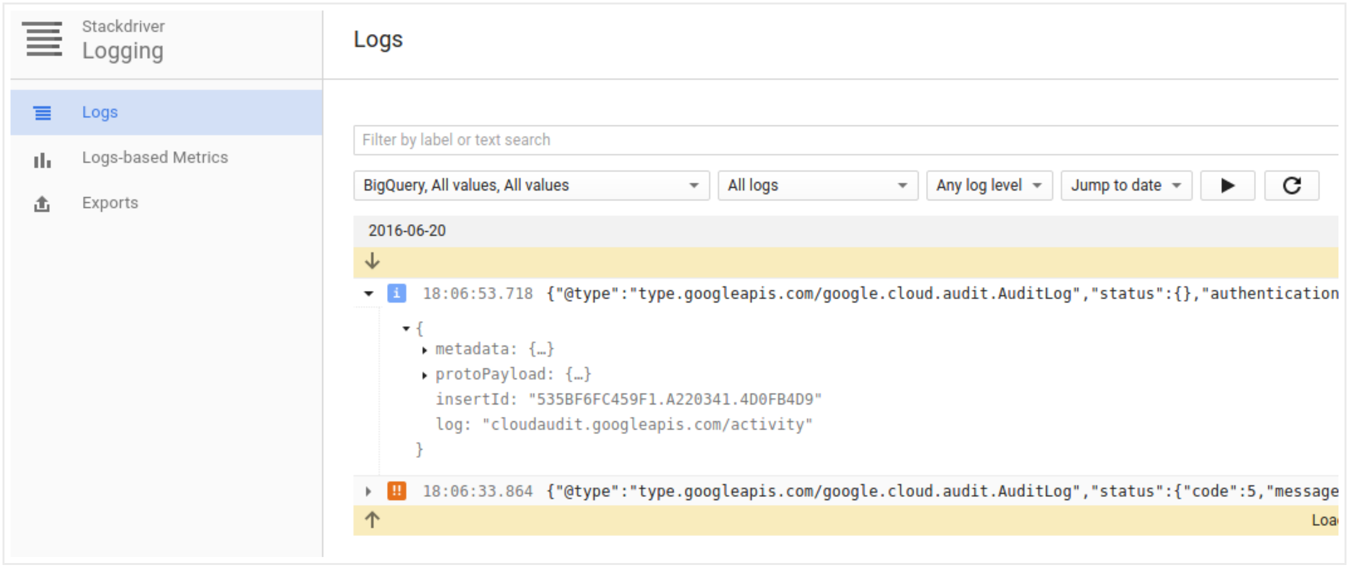
Autoscaling is a feature of managed instance groups (https://cloud.google.com/compute/docs/instance-groups/). A managed instance group is a pool of homogeneous virtual machine instances, created from a common instance template (https://cloud.google.com/compute/docs/instance-templates). An autoscaler adds or remove instances in a managed instance group. Although Compute Engine has both managed and unmanaged instance groups, you can only use managed instance groups with an autoscaler. For more information, see autoscaling on Compute Engine (https://cloud.google.com/compute/docs/autoscaler/).

For an in-depth look at what it takes to build a scalable and resilient web-app solution, see Building scalable and resilient web apps (https://cloud.google.com/solutions/scalable-and-resilient-apps).

Logging and monitoring with Compute Engine

Google Cloud includes features that you can use to keep tabs on what's happening with your website.

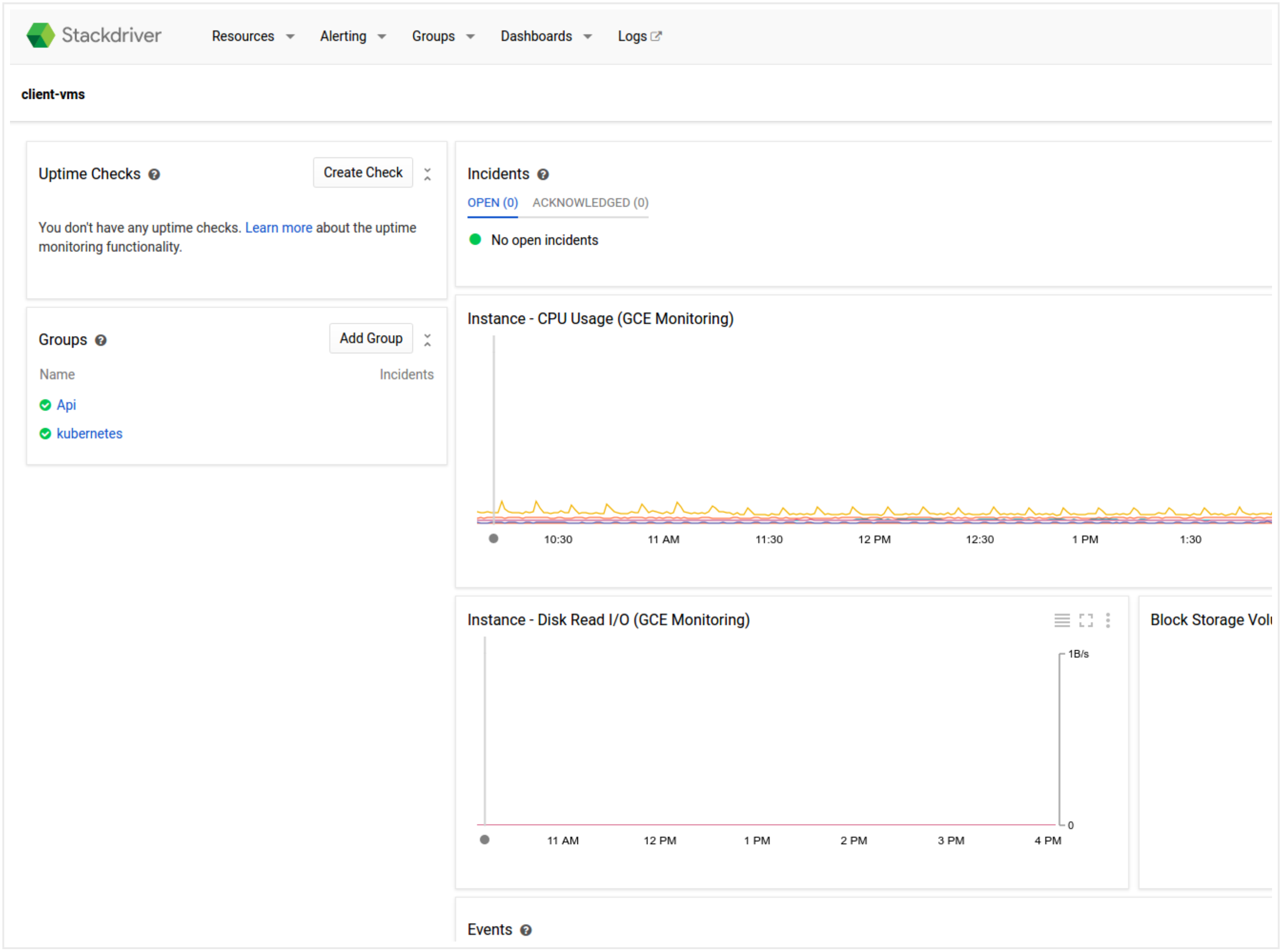
Stackdriver Logging (https://cloud.google.com/logging/docs/) collects and stores logs from apps and services on Google Cloud. You can view or export logs and integrate third-party logs by using a logging agent.



```
18:06:53.718 {"@type":"type.googleapis.com/google.cloud.audit.AuditLog","status":{},"authentication
  {
    metadata: {…}
    protoPayload: {…}
    insertId: "535BF6FC459F1.A220341.4D0FB4D9"
    log: "cloudaudit.googleapis.com/activity"
  }
18:06:33.864 {"@type":"type.googleapis.com/google.cloud.audit.AuditLog","status":{"code":5,"message
```

Stackdriver Monitoring (https://cloud.google.com/monitoring/docs/) provides dashboards and alerts for your site. You configure Monitoring by using the Monitoring Console (https://app.google.stackdriver.com/). You can review performance metrics for cloud services, virtual machines, and common open source servers such as MongoDB, Apache, Nginx, and

Elasticsearch. You can use the Stackdriver Monitoring API to retrieve monitoring data and create custom metrics.



You don't have any uptime checks. Learn more about the uptime monitoring functionality.

No open incidents

Managing DevOps with Compute Engine

For information about managing DevOps with Compute Engine, see the following articles:

- Compute Engine management with Puppet, Chef, Salt, and Ansible (https://cloud.google.com/solutions/google-compute-engine-management-puppet-chef-salt-ansible)
- Automated image builds with Jenkins, Packer, and Kubernetes (https://cloud.google.com/solutions/automated-build-images-with-jenkins-kubernetes)
- Distributed load testing using Kubernetes (https://cloud.google.com/solutions/distributed-load-testing-using-gke)
- Continuous delivery with Travis CI (https://cloud.google.com/solutions/continuous-delivery-with-travis-ci)

- Running Spinnaker on Compute Engine (https://cloud.google.com/solutions/spinnaker-on-compute-engine)
- Managing deployments on Google Cloud with Spinnaker (https://cloud.google.com/solutions/managing-deployments-on-gcp-with-spinnaker)

Using containers with GKE

You might already be using containers, such as Docker (https://docker.io) containers. For web serving, containers offer several advantages, including:

- **Componentization**. You can use containers to separate the various components of your web app. For example, suppose your site runs a web server and a database. You can run these components in separate containers, modifying and updating one component without affecting the other. As your app's design becomes more complex, containers are a good fit for a service-oriented architecture (https://wikipedia.org/wiki/Service-oriented_architecture), including microservices (https://wikipedia.org/wiki/Microservices). This kind of design supports scalability, among other goals.
- **Portability**. A container has everything it needs to run—your app and its dependencies are bundled together. You can run your containers on a variety of platforms, without worrying about the underlying system details.
- **Rapid deployment**. When it's time to deploy, your system is built from a set of definitions and images, so the parts can be deployed quickly, reliably, and automatically. Containers are typically small and deploy much more quickly compared to, for example, virtual machines.

Container computing on Google Cloud offers even more advantages for web serving, including:

- **Orchestration**. GKE (https://cloud.google.com/kubernetes-engine/docs/) is a managed service built on Kubernetes (http://kubernetes.io/), the open source container-orchestration system introduced by Google. With GKE, your code runs in containers that are part of a cluster (https://cloud.google.com/kubernetes-engine/docs/clusters/) that is composed of Compute Engine instances. Instead of administering individual containers or creating and shutting down each container manually, you can automatically manage the cluster through GKE, which uses the configuration you define.

- **Image registration**. Container Registry (https://cloud.google.com/container-registry/docs/) provides private storage for Docker images on Google Cloud. You can access Container Registry through an HTTPS endpoint, so you can pull images from any machine, whether it's a Compute Engine instance or your own hardware. The registry service hosts your custom images in Cloud Storage under your Google Cloud project. This approach ensures by default that your custom images are only accessible by members of your project.
- **Mobility**. This means that you have the flexibility to move and combine workloads with other cloud providers, or mix cloud computing workloads with on-premises implementations to create a hybrid solution.

Storing data with GKE

Because GKE runs on Google Cloud and uses Compute Engine instances as nodes, your storage options have a lot in common with storage on Compute Engine (#gce_storage). You can access Cloud SQL, Cloud Storage, Datastore, and Bigtable through their APIs, or you can use another external storage provider if you choose. However, GKE does interact with Compute Engine persistent disks in a different way than a normal Compute Engine instance would.

A Compute Engine instance includes an attached disk. When you use Compute Engine, as long as the instance exists, the disk volume remains with the instance. You can even detach the disk and use it with a different instance. But in a container, on-disk files are ephemeral. When a container restarts, such as after a crash, the on-disk files are lost. Kubernetes solves this issue by using a volume (https://kubernetes.io/docs/concepts/storage/volumes/) abstraction, and one type of volume is `gcePersistentDisk` (http://kubernetes.io/docs/user-guide/volumes/#gcepersistentdisk). This means that you can use Compute Engine persistent disks with containers to keep your data files from being deleted when you use GKE.

To understand the features and benefits of a volume, you should first understand a bit about pods (https://cloud.google.com/kubernetes-engine/docs/pods/). You can think of a pod as an app-specific logical host for one or more containers. A pod runs on a node instance. When containers are members of a pod, they can share several resources, including a set of shared storage volumes. These volumes enable data to survive container restarts and to be shared among the containers within the pod. Of course, you can use a single container and volume in a pod, too, but the pod is a required abstraction to logically connect these resources to each other.

For an example, see the tutorial Using persistent disks with WordPress and MySQL (https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk/).

Load balancing with GKE

Many large web serving architectures need to have multiple servers running that can share the traffic demands. Because you can create and manage multiple containers, nodes, and pods with GKE, it's a natural fit for a load-balanced web serving system.

Using network load balancing

The easiest way to create a load balancer in GKE is to use Compute Engine's network load balancing (https://cloud.google.com/compute/docs/load-balancing/network/). Network load balancing can balance the load of your systems based on incoming internet protocol data, such as the address, port, and protocol type. Network load balancing uses forwarding rules (https://cloud.google.com/compute/docs/reference/latest/forwardingRules). These rules point to target pools (https://cloud.google.com/compute/docs/reference/latest/targetPools) that list which instances are available to be used for load balancing.

With network load balancing, you can load balance additional TCP/UDP-based protocols such as SMTP traffic, and your app can directly inspect the packets.

You can deploy network load balancing simply by adding the `type: LoadBalancer` field to your service configuration file.

Using HTTP(S) load balancing

If you need more advanced load-balancing features, such as HTTPS load balancing, content-based load balancing, or cross-region load balancing, you can integrate your GKE service with Compute Engine's HTTP/HTTPS load balancing feature. Kubernetes provides the Ingress resource (https://kubernetes.io/docs/concepts/services-networking/ingress/) that encapsulates a collection of rules for routing external traffic to Kubernetes endpoints. In GKE, an Ingress resource handles provisioning and configuring the Compute Engine HTTP/HTTPS load balancer.

For more information about using HTTP/HTTPS load balancing in GKE, see Setting up HTTP load balancing with Ingress (https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer).

Scaling with GKE

For automatic resizing of clusters, you can use the Cluster Autoscaler. This feature periodically checks whether there are any pods that are waiting for a node with free resources but aren't being scheduled. If such pods exist, then the autoscaler resizes the node pool if resizing would allow the waiting pods to be scheduled.

Cluster Autoscaler also monitors the usage of all nodes. If a node isn't needed for an extended period of time, and all of its pods can be scheduled elsewhere, then the node is deleted.

For more information about the Cluster Autoscaler, its limitations, and best practices, see the Cluster Autoscaler documentation (https://cloud.google.com/kubernetes-engine/docs/cluster-autoscaler).

Logging and monitoring with GKE

Like on Compute Engine, Logging (https://cloud.google.com/logging/docs/) and Monitoring (https://cloud.google.com/monitoring/docs/) provide your logging and monitoring services. Logging collects and stores logs from apps and services. You can view or export logs and integrate third-party logs by using a logging agent.

Monitoring provides dashboards and alerts for your site. You configure Monitoring by using the Monitoring Console (https://app.google.stackdriver.com/). You can review performance metrics for cloud services, virtual machines, and common open source servers such as MongoDB, Apache, Nginx, and Elasticsearch. You can use the Monitoring API to retrieve monitoring data and create custom metrics.

Managing DevOps with GKE

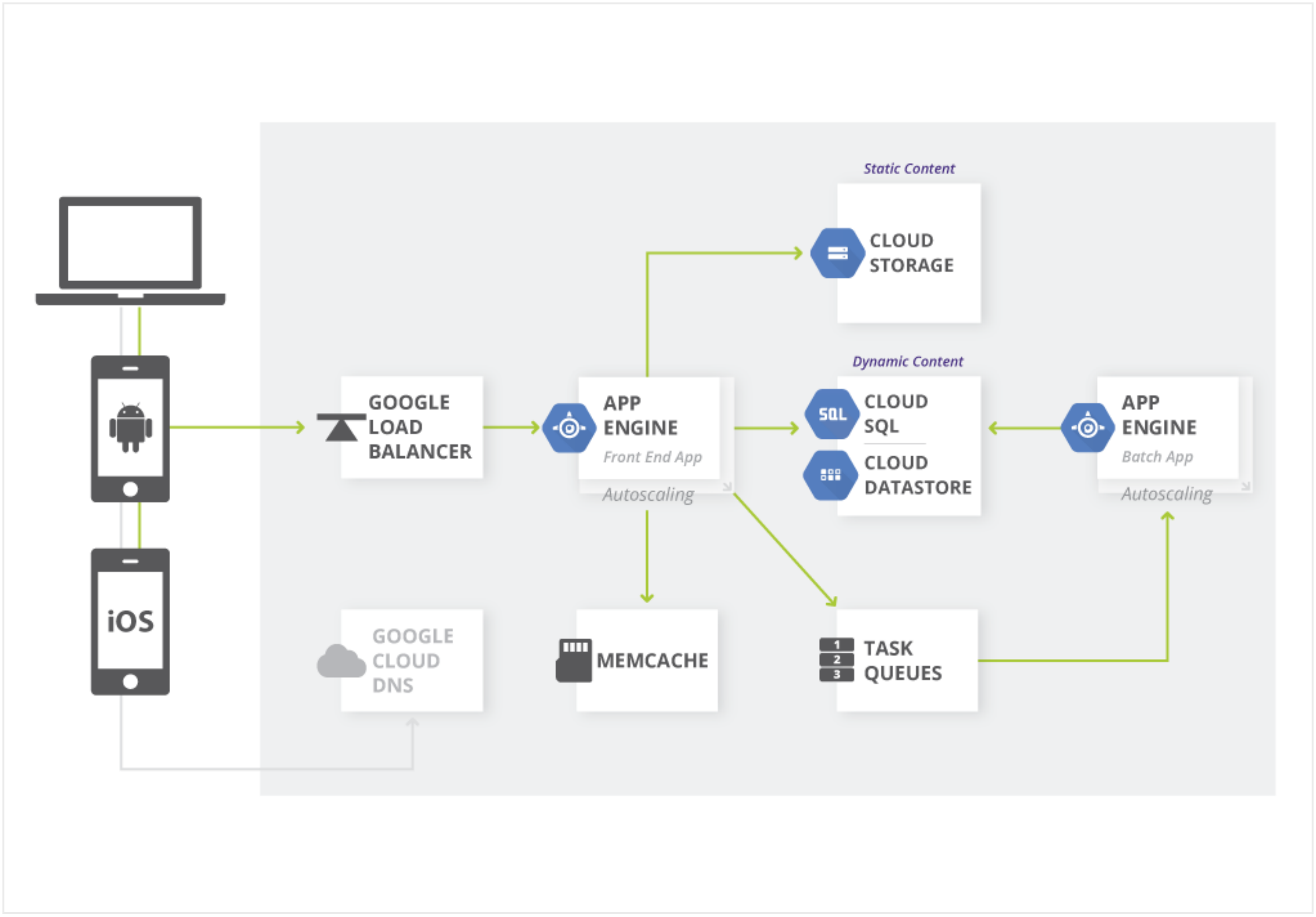
When you use GKE, you're already getting many of the benefits most people think of when they think of DevOps. This is especially true when it comes to ease of packaging, deployment, and management. For your CI/CD workflow needs, you can take advantage of popular tools such as Jenkins. See the following articles:

- Jenkins on GKE (https://cloud.google.com/solutions/jenkins-on-kubernetes-engine)
- Setting up Jenkins on GKE (https://cloud.google.com/solutions/jenkins-on-kubernetes-engine-tutorial)
- Configuring Jenkins for GKE (https://cloud.google.com/solutions/configuring-jenkins-kubernetes-engine)

Building on a managed platform with App Engine

On Google Cloud, the managed platform as a service (PaaS) is called App Engine (https://cloud.google.com/appengine/docs/). When you build your website on App Engine, you get to focus on coding up your features and let Google worry about managing the supporting infrastructure. App Engine provides a wide range of features that make scalability, load balancing, logging, monitoring, and security much easier than if you had to build and manage them yourself. App Engine lets you code in a variety of programming languages, and it can use a variety of other Google Cloud services.

App Engine provides the *standard environment*, which lets you run apps in a secure, sandboxed environment. The App Engine standard environment distributes requests across multiple servers, and scales servers to meet traffic demands. Your app runs in its own secure, reliable environment that's independent of the hardware, operating system, or physical location of the server.



To give you more options, App Engine offers the *flexible environment*. When you use the flexible environment, your app runs on configurable Compute Engine instances, but App Engine manages the hosting environment for you. This means that you can use additional runtimes, including custom runtimes, for more programming language choices. You can also take advantage of some of the flexibility that Compute Engine offers, such as choosing from a variety of CPU and memory options.

Programming languages

The App Engine standard environment provides default runtimes, and you write source code in specific versions of the supported programming languages (https://cloud.google.com/appengine/docs/standard/).

With the flexible environment, you write source code in a version of any of the supported programming languages (https://cloud.google.com/appengine/docs/flexible/). You can customize these runtimes or provide your own runtime with a custom Docker image or Dockerfile.

If the programming language you use is a primary concern, you need to decide whether the runtimes provided by the App Engine standard environment meet your requirements. If they don't, you should consider using the flexible environment.

To determine which environment best meets your app's needs, see Choosing an App Engine environment (https://cloud.google.com/appengine/docs/the-appengine-environments).

Getting started tutorials by language

The following tutorials can help you get started using the App Engine standard environment:

- Hello World in Python (https://cloud.google.com/appengine/docs/python/)
- Hello World in Java (https://cloud.google.com/appengine/docs/java/)
- Hello World in PHP (https://cloud.google.com/appengine/docs/php/)
- Hello World in Go (https://cloud.google.com/appengine/docs/go/)

The following tutorials can help you get started using the flexible environment:

- Getting started with Python (https://cloud.google.com/python/)
- Getting started with Java (https://cloud.google.com/java/)
- Getting started with PHP (https://cloud.google.com/php/)

- Getting started with Go (https://cloud.google.com/go/)
- Getting started with Node.js (https://cloud.google.com/nodejs/)
- Getting started with Ruby (https://cloud.google.com/ruby/)
- Getting started with .NET (https://cloud.google.com/dotnet/)

Storing data with App Engine

App Engine gives you options for storing your data:

Name	Structure	Consistency
Datastore	Schemaless	Strongly consistent except when performing global queries.
Cloud SQL	Relational	Strongly consistent.
Cloud Storage	Files and their associated metadata	Strongly consistent except when performing list operations that get a list of buckets or objects.

You can also use several third-party databases (https://cloud.google.com/appengine/docs/python/using-third-party-databases) with the standard environment.

For more details about storage in App Engine, see Choosing a storage option (https://cloud.google.com/appengine/docs/python/storage), and then select your preferred programming language.

When you use the flexible environment, you can use all of the same storage options as you can with the standard environment, and a wider range of third-party databases as well. For more information about third-party databases in the flexible environment, see Using third-party databases (https://cloud.google.com/appengine/docs/flexible/python/using-third-party-databases).

Load balancing and autoscaling with App Engine

When you build on App Engine, load balancing and autoscaling are automatically managed for you.

Logging and monitoring with App Engine

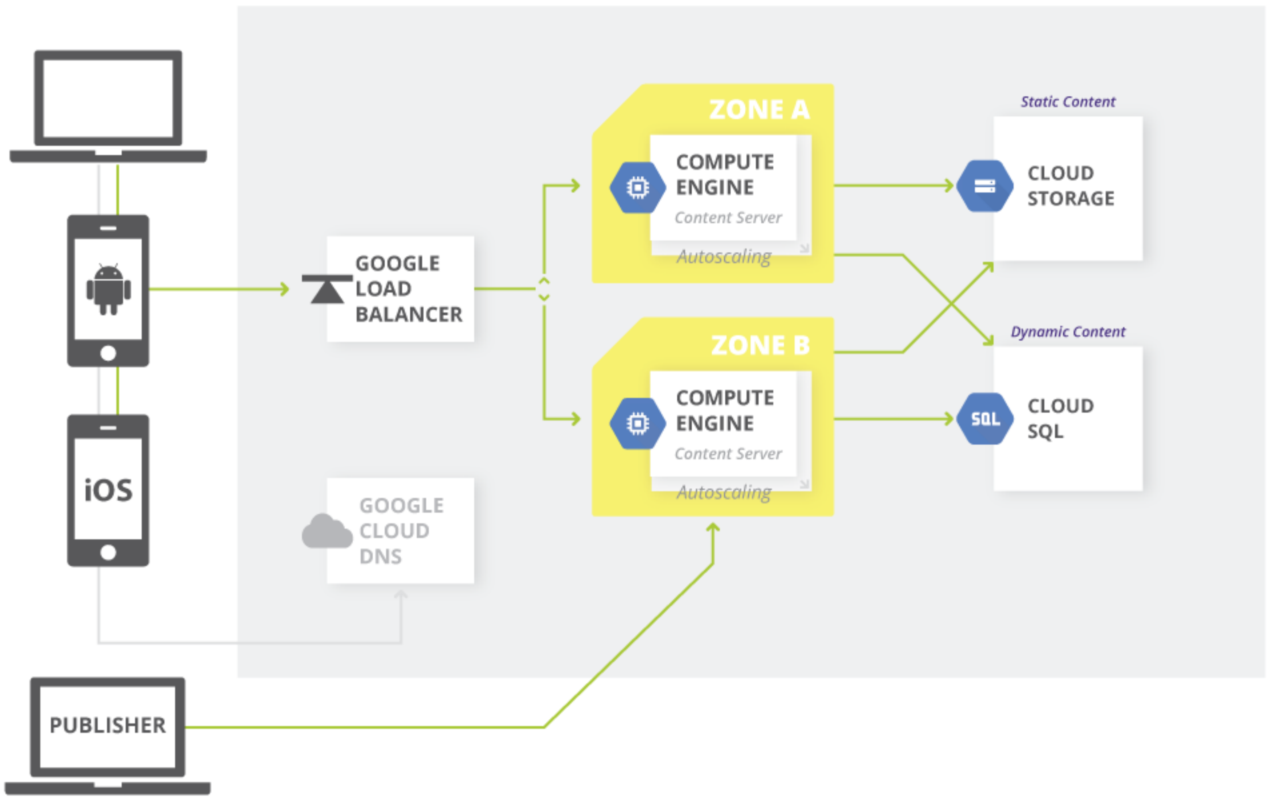
In App Engine, requests are logged automatically, and you can view these logs in the Cloud Console. App Engine also works with standard, language-specific libraries that provide logging functionality and forwards the log entries to the logs in the Cloud Console. For example, in Python (https://cloud.google.com/appengine/docs/python/logs/) you can use the standard Python logging module and in Java (https://cloud.google.com/appengine/docs/java/logs/) you can use the `java.util.logging.Logger` (https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html) API.

Monitoring provides features for monitoring your App Engine apps. Through the Cloud Console, you can monitor incidents, uptime checks, and other details.

Building content management systems

Serving a website means managing your website assets. Cloud Storage provides a global repository for these assets. One common architecture deploys static content to Cloud Storage and then syncs to Compute Engine to render dynamic pages. Cloud Storage works with many third-party content management systems, such as WordPress (https://wordpress.org/plugins/wp2cloud-wordpress-to-cloud/), Drupal (https://www.drupal.org/project/google_cloud_storage), and Joomla (https://www.joomla.org/). Cloud Storage also offers an Amazon S3 compatible API (https://cloud.google.com/storage/docs/interoperability), so any system that works with Amazon S3 can work with Cloud Storage.

For a look at a sample architecture for a content management system, see Content management (https://cloud.google.com/solutions/architecture/contentmanagement).



What's next

- Try out other Google Cloud features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).
