

# GCP Certification Series: 3.2 Deploying and implementing Kubernetes Engine resources



Prashanta Paudel

Oct 25, 2018 · 28 min read

Kubernetes is a hot topic in cloud computing at the moment. Companies are using containers for deployment rather than VM's in the traditional way. It is a high time for kubernetes so let's learn it properly.

***Kubernetes is an open-source container management tool. It does the task of deployment, scaling and load balancing of containers.***

**Kubernetes is not containerization platform** but a multi-container management solution.

Since kubernetes is a container management tool, it is good time know what is a container and what is a docker?

## Containers

containers are a better way to develop and deploy the application.



Containers-symbolic

A container is a packaged software into standardized Units for Development, Shipment and Deployment.

A container is a standard unit of software that packages up the code and all its dependencies so that application runs quickly and reliably from single computing environment.

A container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

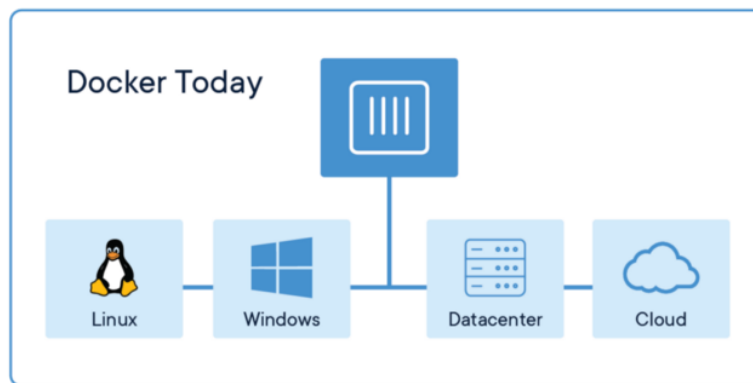


Container

Before containers are deployed they are just container images.  
Container images become container when they run in Docker engine.

Containerized software will always run same regardless of  
infrastructure on which they reside.

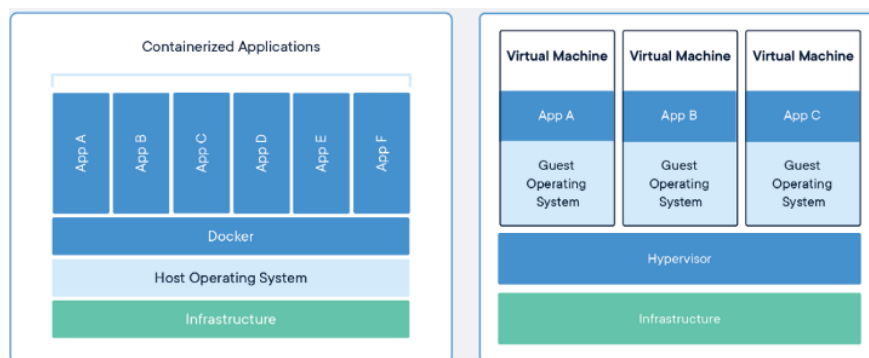
Docker container technology was launched in 2013 as an open source  
Docker Engine. Now it is used everywhere from Linux, windows to  
clouds.



docker containers today

Docker technology is developers and system operators focused as it  
separate dependencies from application and infrastructure.

## Containers and VMs



reference: <https://www.docker.com/resources/what-container>

#### CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

#### VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

reference: <https://www.docker.com/resources/what-container>

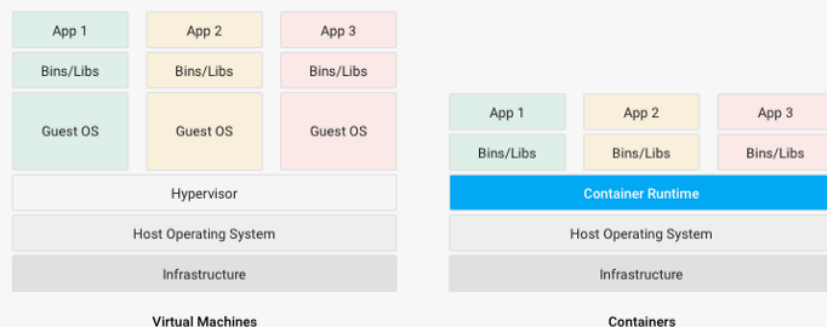
The main difference between containers and VM is Virtual machines virtualize hardware and OS whereas Containers only virtualize OS kernel.

Indeed, few of you know it, but most of you have been using containers for years. Google has its own open-source, container technology **lmctfy** (Let Me Contain That For You). Any time you use some of Google functionality—Search, Gmail, Google Docs, whatever—you're issued a new container.

## Containers 101: What are containers?

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop. Containerization provides a clean separation of concerns, as developers focus on their application logic and dependencies, while IT operations teams can focus on deployment and management without bothering with application details such as specific software versions and configurations specific to the app.

For those coming from virtualized environments, containers are often compared with virtual machines (VMs). You might already be familiar with VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with virtualized access to the underlying hardware. Like virtual machines, containers allow you to package your application together with libraries and other dependencies, providing isolated environments for running your software services. As you'll see below however, the similarities end here as containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.



reference: <https://cloud.google.com/containers/>

## Why Containers?

Instead of virtualizing the hardware stack as with the virtual machines approach, containers virtualize at the operating system level, with multiple containers running atop the OS kernel directly. This means that containers are far more lightweight: they share the OS kernel, start much faster, and use a fraction of the memory compared to booting an entire OS.

There are many container formats available. Docker is a popular, open-source container format that is supported on Google Cloud Platform and by Google Kubernetes Engine.

### Consistent Environment

Containers give developers the ability to create predictable environments that are isolated from other applications. Containers can also include software dependencies needed by the application, such as specific versions of programming language runtimes and other software libraries. From the developer's perspective, all this is guaranteed to be consistent no matter where the application is ultimately deployed. All this translates to productivity: developers and IT Ops teams spend less time debugging and diagnosing differences in environments, and more time shipping new functionality for users. And it means fewer bugs since developers can now make assumptions in dev and test environments they can be sure will hold true in production.

### Run Anywhere

Containers are able to run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal; on a developer's machine or in data centers on-premises; and of course, in the public cloud. The widespread popularity of the [Docker image format](#) for containers further helps with portability. Wherever you want to run your software, you can use containers.

### Isolation

Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications.

### Why Sandbox anyway?

Containers silo applications from each other unless you explicitly connect them. That means you don't have to worry about conflicting dependencies or resource contention — you set explicit resource limits for each service. Importantly, it's an additional layer of security since your applications aren't running directly on the host operating system.

reference: <https://cloud.google.com/containers/>

## From Code to Applications

Containers allow you to package your application and its dependencies together into one succinct manifest that can be version controlled, allowing for easy replication of your application across developers on your team and machines in your cluster.

Just as how software libraries package bits of code together, allowing developers to abstract away logic like user authentication and session management, containers allow your application as a whole to be packaged, abstracting away the operating system, the machine, and even the code itself. Combined with a service-based architecture, the entire unit that developers are asked to reason about becomes much smaller, leading to greater agility and productivity. All this eases development, testing, deployment, and overall management of your applications.

### Monolithic to Service Based Architecture

Containers work best for service based architectures. Opposed to monolithic architectures, where every pieces of the application is intertwined — from IO to data processing to rendering — service based architectures separate these into separate components. Separation and division of labor allows your services to continue running even if others are failing, keeping your application as a whole more reliable.

Componentization also allows you to develop faster and more reliably; smaller codebases are easier to maintain and since the services are separate, it's easy to test specific inputs for outputs.

Containers are perfect for service based applications since you can health check each container, limit each service to specific resources and start and stop them independently of each other.

And since containers abstract the code away, containers allow you to treat separate services as black boxes, further decreasing the space a developer needs to be concerned with. When developers work on services that depends on another, they can easily start up a container for that specific service without having to waste time setting up the correct environment and troubleshooting beforehand.


reference: reference: <https://cloud.google.com/containers/>

## Docker

Docker provides freedom of infrastructure for developers and IT to build, manage and secure business-critical applications without the fear of technology or infrastructure lock-in.

*Docker is also an organization that developed docker containers technology.*


### How Docker Works for You



**Developers**

Tooling that is simple to use, yet powerful and delivers a great user experience so you can focus on what you love — writing great code.


[→ Get Started](#)



**IT Operations**

Docker delivers an enterprise-ready container platform to deploy and run applications in a way that makes the best sense for your customers and business.

[→ Get Started](#)



**Business Leader**

Docker provides a platform to drive your digital transformation by accelerating new innovation while dramatically driving down your existing IT costs.

[→ Get Started](#)

reference: <https://www.docker.com/why-docker>

Docker containers are the fastest growing cloud-enabling technology and driving a new era of computing and application architecture with their lightweight approach to bundling applications and dependencies into isolated, yet highly portable application packages.

Containers alone are not enough to provide value at the scale of your enterprise and do not directly address the compliance, security and operational needs of your organization.

Why use Docker Containers?

- VM's are bulky in system requirement and needs more resources but containers don't require that much of resources.
- Docker containers support Continuous integration/ Continuous deployment(CI/CD) that encourage developers to integrate their code into shared repository early.
- Docker containers are easy to deploy in the cloud.
- It easily incorporates in most DevOps applications, including Puppet, Chef, Vagrant, and Ansible.
- Jenkins an open-source CI/CD program, to automate the delivery of new software in containers.
- 10's or 100's of containers for load balancing the traffic and ensuring high availability.

## Container Orchestration

Container orchestration is the process of managing containers in the cloud.

Major container orchestration tools are

- Docker Swarm
- Kubernetes
- Mesosphere

## What Is Kubernetes | Kubernetes Introducti...



### DRAWBACKS

This, in turn, means one thing VM hypervisors can do that container can't is to use different operating systems or kernels. So, for example, you can use Microsoft Azure to run both instances of Windows Server 2012 and SUSE Linux Enterprise Server, at the same time. With Docker, all containers must use the same operating system and kernel.

On the other hand, if all you want to do is get the most server application instances running on the least amount of hardware, you couldn't care less about running multiple operating system VMs. If multiple copies of the same application are what you want, then you'll love containers.

## Kubernetes



reference: Wikipedia



Kubernetes is an open source container orchestration system for automating deployment, scaling, and management of containerized applications.

It was originally designed by Google and is now maintained by Cloud Native Computing Foundation. It aims to provide a “platform for automating deployment, scaling, and operations of application containers across the cluster of hosts.

## History

Kubernetes was founded by Joe Beda, Brendan Burns and Craig McLuckie, was quickly joined by other Google engineers including Brian Grant and Tim Hockin and was **first announced by Google in mid-2014**. Its development and design are heavily **influenced by Google’s Borg system** and many of the top contributors to the project previously worked on Borg. The original codename for Kubernetes within Google was Project Seven, a reference to Star Trek character Seven of Nine that is a ‘friendlier’ Borg. The seven spokes on the wheel of the Kubernetes logo are a nod to that codename.

**Kubernetes v1.0 was released on July 21, 2015.** Along with the Kubernetes v1.0 release, **Google partnered with the Linux Foundation to form the Cloud Native Computing Foundation (CNCF)** and offered Kubernetes as a seed technology. On March 6, 2018, Kubernetes Project reached ninth place in commits at GitHub, and second place in authors and issues, just below Linux Project.

## What can Kubernetes do for you?

With modern web services, users expect applications to be available 24/7, and developers expect to deploy new versions of those applications several times a day. Containerization helps package software to serve these goals, enabling applications to be released and updated in an easy and fast way without downtime. Kubernetes helps you make sure those containerized applications run where and when you want and helps them find the resources and tools they need to work. Kubernetes is a production-ready, open source platform designed with Google’s accumulated experience in container orchestration, combined with best-of-breed ideas from the community.

# Kubernetes Clusters

**Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.** The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized. Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. **Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.** Kubernetes is an open-source platform and is production-ready.

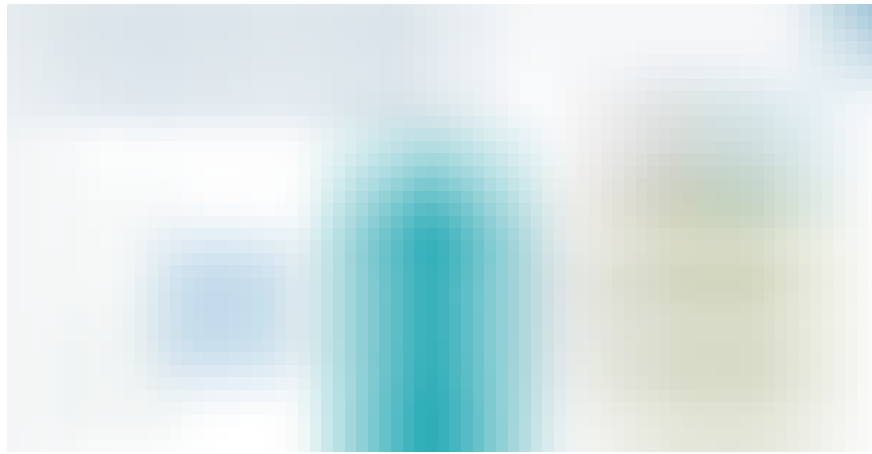
A Kubernetes cluster consists of two types of resources:

- The **Master** coordinates the cluster
- **Nodes** are the workers that run applications

**The Master is responsible for managing the cluster.** The master coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

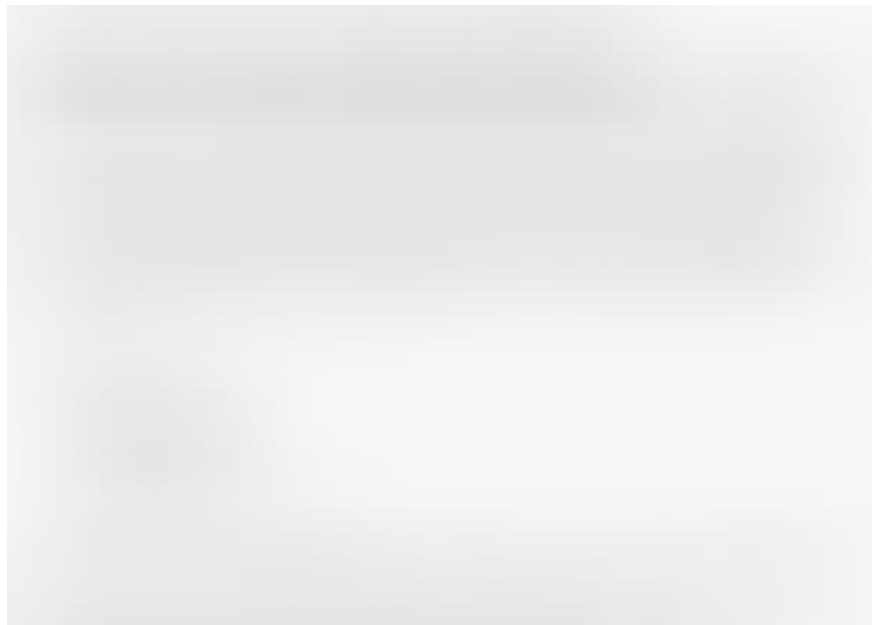
**A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.** Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes master. The node should also have tools for handling container operations, such as Docker or rkt. A Kubernetes cluster that handles production traffic should have a minimum of three nodes.

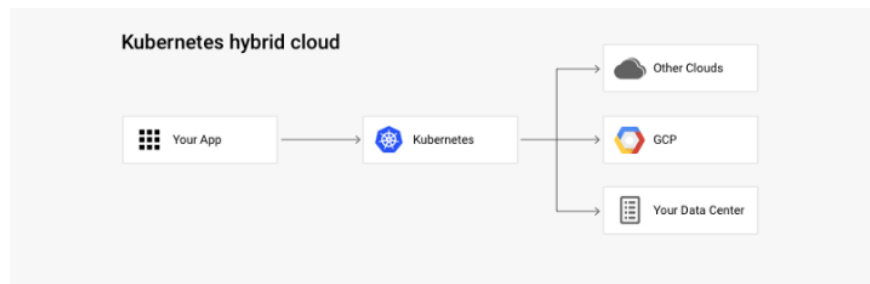
*Masters manage the cluster and the nodes are used to host the running applications.*



reference:<https://dzone.com/articles/what-is-kubernetes-container-orchestration-tool>

When you deploy applications on Kubernetes, you tell the master to start the application containers. The master schedules the containers to run on the cluster's nodes. **The nodes communicate with the master using the Kubernetes API**, which the master exposes. End users can also use the Kubernetes API directly to interact with the cluster.





## Your Cluster on Google

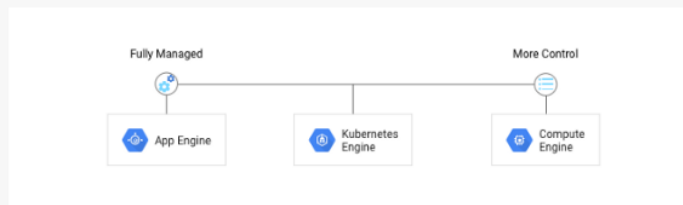
Of course, Kubernetes runs best on Google Cloud Platform. [Google Kubernetes Engine](#) is the premier managed Kubernetes solution that gets you quickly set up and production-ready.

Kubernetes Engine is fully managed by Google reliability engineers, the ones who know containers the best, ensuring your cluster is highly available and up-to-date. It integrates seamlessly with all GCP services, such as [Stackdriver](#) monitoring, diagnostics, and logging; [Identity and Access Management](#); and Google's best-in-class [networking](#) infrastructure.

### Kubernetes Engine Features

- ✓ Managed open-source Kubernetes
- ✓ 99.5% SLA, and high availability with integrated multi-zone deployments
- ✓ Seamless integration of other GCP services
- ✓ Industry leading price per performance
- ✓ Flexible & interoperable with your on-premises clusters or other cloud providers
- ✓ Google-grade managed-infrastructure

But we love to give you options. Google Cloud Platform offers you a full spectrum for running your containers. From fully managed platform-as-a-service with [Google App Engine Flexible Environment](#) to cluster management with Kubernetes Engine to roll-it-yourself infrastructure on world-class price-to-performance [Google Compute Engine](#), you can find your ideal solution for running containers on Google Cloud Platform.



reference: <https://cloud.google.com/containers/>

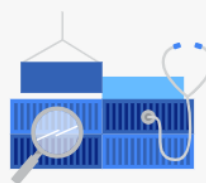


## Deploy a Wide Variety of Applications

Kubernetes Engine enables rapid application development and iteration by making it easy to deploy, update, and manage your applications and services. Kubernetes Engine isn't just for stateless applications either; you can attach persistent storage, and even run a database in your cluster. Simply describe the compute, memory, and storage resources your application containers require, and Kubernetes Engine provisions and manages the underlying cloud resources automatically. Support for hardware accelerators makes it easy to run Machine Learning, General Purpose GPU, High-Performance Computing, and other workloads that benefit from specialized hardware accelerators.

## Operate Seamlessly with High Availability

Control your environment from the built-in Kubernetes Engine dashboard in Google Cloud console. Use routine health checks to detect and replace hung, or crashed, applications inside your deployments. Container replication strategies, monitoring, and automated repairs help ensure that your services are highly available and offer a seamless experience to your users. Google Site Reliability Engineers (SREs) constantly monitor your cluster and its compute, networking, and storage resources so you don't have to, giving you back time to focus on your applications.



<https://cloud.google.com/kubernetes-engine/>

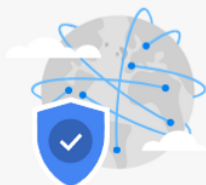



## Scale Effortlessly to Meet Demand

Go from a single machine to thousands: Kubernetes Engine autoscaling allows you to handle increased user demand for your services, keeping them available when it matters most. Then, scale back in the quiet periods to save money, or schedule low-priority batch jobs to use up spare cycles. Kubernetes Engine helps you get the most out of your resource pool.

## Run Securely on Google's Network

Connect to and isolate clusters no matter where you are with fine-grained network policies using Global Virtual Private Cloud (VPC) in Google Cloud. Use public services behind a single global anycast IP address for seamless load balancing. Protect against DOS and other types of edge attacks on your containers.





### Move Freely between On-premises and Clouds

Kubernetes Engine runs Certified Kubernetes ensuring portability across clouds and on-premises. There's no vendor lock-in: you're free to take your applications out of Kubernetes Engine and run them anywhere Kubernetes is supported, including on your own on-premises servers. You can tailor integrations such as monitoring, logging, and CI/CD using Google Cloud Platform (GCP) and third party solutions in the ecosystem.

## KUBERNETES ENGINE FEATURES

Run containerized applications on production-ready Kubernetes, managed by Google Cloud.

<b>Identity &amp; Access Management</b> Control access in the cluster with your Google accounts and role permissions.	<b>Stateful Application Support</b> Kubernetes Engine isn't just for 12-factor apps. You can attach persistent storage to containers, and even host complete databases.
<b>Hybrid Networking</b> Reserve an IP address range for your cluster, allowing your cluster IPs to coexist with private network IPs via <a href="#">Google Cloud VPN</a> .	<b>Docker Image Support</b> Kubernetes Engine supports the common Docker container format.
<b>Security and Compliance</b> Kubernetes Engine is backed by Google security team of over 750 experts and is both HIPAA and PCI DSS 3.1 compliant.	<b>Fully Managed</b> Kubernetes Engine clusters are fully managed by Google Site Reliability Engineers (SREs), ensuring your cluster is available and up-to-date.
<b>Integrated Logging &amp; Monitoring</b> Enable <a href="#">Stackdriver Logging</a> and <a href="#">Stackdriver Monitoring</a> with simple checkbox configurations, making it easy to gain insight into how your application is running.	<b>OS Built for Containers</b> Kubernetes Engine runs on <a href="#">Container-Optimized OS</a> , a hardened OS built and managed by Google.
<b>Auto Scale</b> Automatically scale your application deployment up and down based on resource utilization (CPU, memory).	<b>Private Container Registry</b> Integrating with <a href="#">Google Container Registry</a> makes it easy to store and access your private Docker images.
<b>Auto Upgrade</b> Automatically keep your cluster up to date with the latest release version of Kubernetes. Kubernetes release updates are quickly made available within Kubernetes Engine.	<b>Fast Consistent Builds</b> Use <a href="#">Google Cloud Build</a> to reliably deploy your containers on Kubernetes Engine without needing to setup authentication.
<b>Auto Repair</b> When auto repair is enabled, if a node fails a health check Kubernetes Engine initiates a repair process for that node.	<b>Workload Portability, on-premises and cloud</b> Kubernetes Engine runs Certified Kubernetes, enabling workload portability to other Kubernetes platforms across clouds and on-premises.
<b>Resource Limits</b> Kubernetes allows you to specify how much CPU and memory (RAM) each Container needs, which is used to better organize workloads within your cluster.	<b>GPU support<sup>BETA</sup></b> Kubernetes Engine supports GPU and makes it easy to run ML, GPGPU, HPC, and other workloads that benefit from specialized hardware accelerators.

Reference: <https://cloud.google.com/kubernetes-engine/>

## Small Tutorial in CLI for kubernets deployment

Welcome to the CI/CD with GKE tutorial terminal emulator!

In this tutorial, you will create a Kubernetes cluster on GKE, spin up a deployment from Google Container Registry, scale up the deployment and update the code base.

### STEP 1: CREATE A CLUSTER

Enter the following command to create a cluster:

```

gcloud container clusters create myCluster
$ gcloud container clusters create mycluster
Creating cluster mycluster...done.
Created [https://container.googleapis.com/v1/projects/kubernetes-
terminal-simula
tor/zones/us-west1-a/clusters/mycluster].
kubeconfig entry generated for mycluster.
NAME ZONE MASTER_VERSION MASTER_IP MACHINE_TYPE
NODE_VERSION N
UM_NODES STATUS
mycluster us-west1-a 1.7.8-gke.0 198.51.100.41 n1-standard-1 1.7.8-
gke
.0 3 RUNNING

```

#### STEP 2: SPIN UP A DEPLOYMENT FROM GOOGLE CONTAINER REGISTRY.

Now let's pull and execute a Kubernetes deployment from Google Container Register

y:

```

kubectl run app—image gcr.io/google-samples/hello-app:1.0
$ kubectl run app—image gcr.io/google-samples/hello-app:1.0
deployment “app” created

```

#### STEP 3: SCALE DEPLOYMENT.

We should scale up the deployment across three pods for redundancy:

```

kubectl scale deployment app—replicas 3
$ kubectl scale deployment app—replicas 3
deployment “app” scaled

```

#### STEP 4: OPEN PORTS.

Now that the website is deployed, we are ready to open it up to the world:

```

kubectl expose deployment app—port 80—type=LoadBalancer
$ kubectl expose deployment app—port 80—type=Loadbalancer
service “app” exposed

```

#### STEP 5: CONFIRM DEPLOYMENT.

Confirm service deployment:

```

kubectl get service app
$ kubectl get service app
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

```

```
app undefined 203.0.113.51 203.0.113.16 80—type:31834/TCP 1m
```

#### STEP 6: CONFIRM AVAILABILITY.

Confirm service is deployed and available:

```
curl http://203.0.113.16:80--type
$ curl http://203.0.113.16:80--type
Hello, world!
Version: 1.0.0
Hostname: app-970732273-p6j37
```

#### STEP 7: UPDATE CODEBASE.

Let's imagine that the website has been updated and we need to deploy the update

d codebase to production—it's easy, just enter the following command:

```
kubectl set image deployment app app=gcr.io/google-samples/hello-
app:2.0
$ kubectl set image deployment app app=gcr.io/google-samples/hello-
app:2.0
deployment "app" image updated
```

#### STEP 8: CONFIRM UPDATE.

Congratulations, your new website is up and running and automatically deployed t

o all running pods! Let's confirm:

```
curl http://203.0.113.16:80--type
$ curl http://203.0.113.16:80--type
Hello, world!
Version: 2.0.0
Hostname: app-970732273-hnhzc
```

TUTORIAL COMPLETE!

## GKE Overview

Google's Kubernetes Engine provides a managed environment for deploying, managing and scaling your containerized application using Google infrastructure.

The environment generally consists of Google compute engine instances grouped together to form a cluster.



## Cluster Orchestration with GKE

GKE clusters are powered by the Kubernetes open source cluster management system. Kubernetes provides the mechanisms through which you interact with your cluster. You use Kubernetes commands and resources to deploy and manage your applications, perform administration tasks and set policies, and monitor the health of your deployed workloads.

## Kubernetes on Google Cloud Platform

When you run a GKE cluster, you also gain the benefit of advanced cluster management features that Google Cloud Platform provides. These include:

- Google Cloud Platform's load-balancing for Compute Engine instances
- Node pools to designate subsets of nodes within a cluster for additional flexibility
- Automatic scaling of your cluster's node instance count
- Automatic upgrades for your cluster's node software
- Node auto-repair to maintain node health and availability
- Logging and Monitoring with Stackdriver for visibility into your cluster

## Kubernetes Versions and Features

GKE cluster masters are automatically upgraded to run new versions of Kubernetes as those versions become stable, so you can take advantage of newer features from the open-source Kubernetes project.

## GKE Workloads

GKE works with containerized applications: applications packaged into hardware independent, isolated user-space instances, for example by using Docker. In GKE and Kubernetes, these containers, whether for applications or batch jobs, are collectively called *workloads*. Before you

deploy a workload on a GKE cluster, you must first package the workload into a container.

Google Cloud Platform provides continuous integration and continuous delivery tools to help you build and serve application containers. You can use Google Cloud Build to build container images (such as Docker) from a variety of source code repositories, and Google Container Registry to store and serve your container images.

## Cluster Architecture

In Google Kubernetes Engine, a *cluster* consists of at least one *cluster master* and multiple worker machines called *nodes*. These master and node machines run the Kubernetes cluster orchestration system.

A cluster is the foundation of GKE: the Kubernetes objects that represent your containerized applications all run on top of a cluster.

## Cluster master

The *cluster master* runs the Kubernetes control plane processes, including the Kubernetes API server, scheduler, and core resource controllers. The master's lifecycle is managed by GKE when you create or delete a cluster. This includes upgrades to the Kubernetes version running on the cluster master, which GKE performs automatically, or manually at your request if you prefer to upgrade earlier than the automatic schedule.

## Cluster master and the Kubernetes API

The master is the unified endpoint for your cluster. All interactions with the cluster are done via Kubernetes API calls, and the master runs the **Kubernetes API Server** process to handle those requests. You can make Kubernetes API calls directly via HTTP/gRPC, or indirectly, by running commands from the Kubernetes command-line client (`kubectl`) or interacting with the UI in the GCP Console.

The cluster master's API server process is the hub for all communication for the cluster. All internal cluster processes (such as the cluster nodes, system, and components, application controllers) all act as clients of

the API server; the API server is the single “source of truth” for the entire cluster.

## Master and node interaction

The cluster master is responsible for deciding what runs on all of the cluster’s nodes. This can include scheduling workloads, like containerized applications, and managing the workloads’ lifecycle, scaling, and upgrades. The master also manages network and storage resources for those workloads.

The master and nodes also communicate using Kubernetes APIs.

## Nodes

A cluster typically has one or more *nodes*, which are the worker machines that run your containerized applications and other workloads. The individual machines are Compute Engine VM instances that GKE creates on your behalf when you create a cluster.

Each node is managed from the master, which receives updates on each node’s self-reported status. You can exercise some manual control over node lifecycle, or you can have GKE perform automatic repairs and automatic upgrades on your cluster’s nodes.

A node runs the services necessary to support the Docker containers that make up your cluster’s workloads. These include the Docker runtime and the Kubernetes node agent ( `kubelet` ) which communicates with the master and is responsible for starting and running Docker containers scheduled on that node.

In GKE, there are also a number of special containers that run as per-node agents to provide functionality such as log collection and intra-cluster network connectivity.

## Node machine type

Each node is of a standard Compute Engine machine type. The default type is `n1-standard-1` , with 1 virtual CPU and 3.75 GB of memory. You can select a different machine type when you create a cluster.

## Node OS images

Each node runs a specialized OS image for running your containers. You can specify which OS image your clusters and node pools use.

## Minimum CPU platform

When you create a cluster or node pool, you can specify a baseline minimum CPU platform for its nodes. Choosing a specific CPU platform can be advantageous for advanced or compute-intensive workloads. For more information, refer to the Minimum CPU Platform.

## Pod

### What is a Pod?

**Pods** are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster.

Pods contain one or more *containers*, such as Docker containers. When a Pod runs multiple containers, the containers are managed as a single entity and share the Pod's resources. Generally, running multiple containers in a single Pod is an advanced use case.

Pods also contain shared networking and storage resources for their containers:

- **Network:** Pods are automatically assigned unique IP addresses. Pod containers share the same network namespace, including IP address and network ports. Containers in a Pod communicate with each other inside the Pod on `localhost`.
- **Storage:** Pods can specify a set of shared storage volumes that can be shared among the containers.

Pods run on **nodes** in your cluster. Once created, a Pod remains on its node until its process is complete, the Pod is deleted, the Pod is **evicted** from the node due to lack of resources, or the node fails. If a node fails, Pods on the node are automatically scheduled for deletion.

## Pod lifecycle

Pods are ephemeral. They are not designed to run forever, and when a Pod is terminated it cannot be brought back. In general, Pods do not disappear until they are deleted by a user or by a controller.

Pods do not “heal” or repair themselves. For example, if a Pod is scheduled on a node which later fails, the Pod is deleted. Similarly, if a Pod is evicted from a node for any reason, the Pod does not replace itself.

Each Pod has a `PodStatus` API object, which is represented by a Pod's `status` field. Pods publish their *phase* to the `status: phase` field. The phase of a Pod is a high-level summary of the Pod in its current state.

When you run `kubectl get pod` to inspect a Pod running on your cluster, a Pod can be in one of the following possible phases:

- **Pending:** Pod has been created and accepted by the cluster, but one or more of its containers are not yet running. This phase includes time spent being scheduled on a node and downloading images.
- **Running:** Pod has been bound to a node, and all of the containers have been created. At least one container is running, is in the process of starting, or is restarting.
- **Succeeded:** All containers in the Pod have terminated successfully. Terminated Pods do not restart.
- **Failed:** All containers in the Pod have terminated, and at least one container has terminated in failure. A container “fails” if it exits with a non-zero status.
- **Unknown:** The state of the Pod cannot be determined.

## Pod usage patterns

Pods can be used in two main ways:

- **Pods that run a single container.** The simplest and most common Pod pattern is a single container per pod, where the

single container represents an entire application. In this case, you can think of a Pod as a wrapper.

- **Pods that run multiple containers that need to work together.**  
Pods with multiple containers are primarily used to support co-located, co-managed programs that need to share resources. These co-located containers might form a single cohesive unit of service—one container serving files from a shared volume while another container refreshes or updates those files. The Pod wraps these containers and storage resources together as a single manageable entity.

Each Pod is meant to run a single instance of a given application. If you want to run multiple instances, you should use one Pod for each instance of the application. This is generally referred to as *replication*. Replicated Pods are created and managed as a group by a controller, such as a Deployment.

## Pod termination

Pods terminate gracefully when their processes are complete. By default, all terminations are graceful within 30 seconds.

You can manually delete a Pod using the `kubectl delete` command. The command's `--grace-period` flag allows you to override the default grace period.

## Deployment

This page describes Kubernetes Deployment objects and their use in Google Kubernetes Engine.

## What is a Deployment?

*Deployments* represent a set of multiple, identical Pods with no unique identities. A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. In this way, Deployments help ensure that one or more instances of your application are available to serve user requests. Deployments are managed by the Kubernetes Deployment controller.

Deployments use a Pod template, which contains a specification for its Pods. The Pod specification determines how each Pod should look like: what applications should run inside its containers, which volumes the Pods should mount, its labels, and more.

When a Deployment's Pod template is changed, new Pods are automatically created one at a time.

## Usage patterns

Deployments are well-suited for stateless applications that use `ReadOnlyMany` or `ReadWriteMany` volumes mounted on multiple replicas, but are not well-suited for workloads that use `ReadWriteOnce` volumes. For stateful applications using `ReadWriteOnce` volumes, use `StatefulSets`. `StatefulSets` are designed to deploy stateful applications and clustered applications that save data to persistent storage, such as Compute Engine persistent disks. `StatefulSets` are suitable for deploying Kafka, MySQL, Redis, ZooKeeper, and other applications needing unique, persistent identities and stable hostnames.

## Creating Deployments

You can create a Deployment using the `kubectl run`, `kubectl apply`, or `kubectl create` commands.

Once created, the Deployment ensures that the desired number of Pods are running and available at all times. The Deployment automatically replaces Pods that fail or are evicted from their nodes.

The following is an example of a Deployment manifest file in YAML format:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

```
    app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

In this example:

- A Deployment named `nginx` is created, indicated by the `metadata: name` field.
- The Deployment creates three replicated Pods, indicated by the `replicas` field.
- The Pod template, or `spec: template` field, indicates that its Pods are labelled `app: nginx`.
- The Pod template's specification, or `template: spec` field, indicates that the Pods run one container, `nginx`, which runs the `nginx` Docker Hub image at version 1.7.9.
- The Deployment opens port 80 for use by the Pods.

In sum, the Pod template contains the following instructions for Pods created by this Deployment:

- Each Pod is labeled.
- Create one container and name it `nginx`.
- Run the `nginx` image at version `1.7.9`.
- Open a port `80` to send and receive traffic.

For more information about creating Deployments, refer to [Creating a Deployment](#).

## Updating Deployments

You can update a Deployment by making changes to the Deployment's Pod template specification. Making changes to the `specification` field automatically triggers an update rollout. You can use `kubectl`, the



Kubernetes API, or the GKE Workloads menu in Google Cloud Platform Console.

By default, when a Deployment triggers an update, the Deployment stops the Pods, gradually scales down the number of Pods to zero, then drains and terminates the Pods. Then, the Deployment uses the updated Pod template to bring up new Pods.

Old Pods are not removed until a sufficient number of new Pods are Running, and new Pods are not created until a sufficient number of old Pods have been removed. To see in which order Pods are brought up and are removed, you can run `kubectl describe deployments`.

Deployments can ensure that at least one less than the desired number of replicas are running, with at most one Pod being unavailable. Similarly, Deployments can ensure that at least one more than the desired number of replicas, with at most one more Pod than the desired running.

You can roll back an update using the `kubectl rollout undo` command. You can also use `kubectl rollout pause` to temporarily halt a Deployment.

## Managing Deployments

The following is a list of common management tasks for Deployments:

- Inspect a Deployment
- Scale a Deployment
- Autoscale a Deployment using a `HorizontalPodAutoscaler` object
- Delete a Deployment

## Status and lifecycle

Deployments can be in one of three states during its lifecycle: progressing, completed, or failed.

A *progressing* state indicates that the Deployment is in process of performing its tasks, like bringing up or scaling its Pods.

A *completed* state indicates that the Deployment has successfully completed its tasks, all of its Pods are running with the latest specification and are available, and no old Pods are still running.

A *failed* state indicates that the Deployment has encountered one or more issues that prevent it from completing its tasks. Some causes include insufficient quotas or permissions, image pull errors, limit ranges, or runtime errors. To investigate what causes a Deployment to fail, you can run `kubectl get deployment [DEPLOYMENT+NAME] -o yaml` and examine the messages in the `status: conditions` field.

You can monitor a Deployment's progress or check its status using the `kubectl rollout status` command.

---

## Deploying a Kubernetes Engine cluster

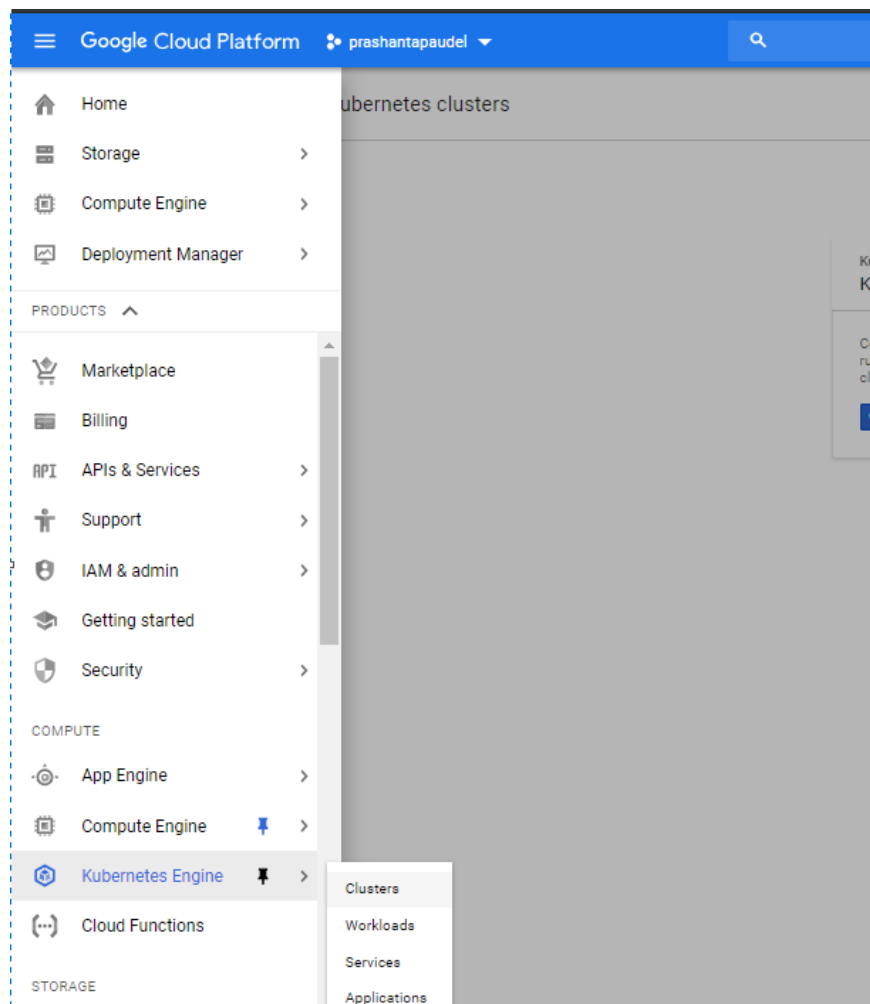
### Objectives

To package and deploy your application on GKE, you must:

1. Package your app into a Docker image
2. Run the container locally on your machine (optional)
3. Upload the image to a registry
4. Create a container cluster
5. Deploy your app to the cluster
6. Expose your app to the Internet
7. Scale up your deployment
8. Deploy a new version of your app

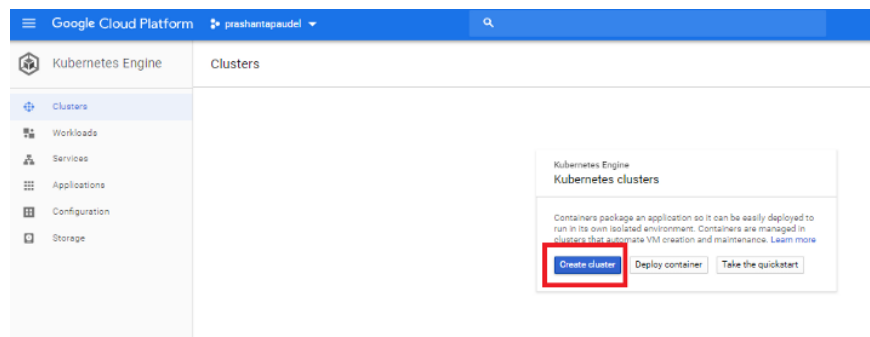
| *From GCP Console*

First, go to Google Console and click Compute engine then go to Kubernetes Dashboard



Kubernetes Dashboard

Now you will land in the page of cluster inside kubernetes Engine



Cluster page

Now, click on Create Cluster

✕ Create a Kubernetes cluster

**Cluster templates**

Select a template with preconfigured settings, or customize a template to suit your needs.

- ☐ Clone an existing cluster  
Select one of your existing clusters to populate fields.
- ☒ **Standard cluster**  
Continuous integration, web serving, backends. Best choice for further customization or if you are not sure what to choose.
- ☐ Your first cluster  
Experimenting with Kubernetes Engine, deploying your first application. Affordable choice to get started.
- ☐ CPU intensive applications  
Web crawling or anything else that requires more CPU.
- ☐ Memory intensive applications  
Databases, analytics, things like Hadoop, Spark, ETL, or anything else that requires more memory.
- ☐ GPU Accelerated Computing  
Machine learning, video transcoding, scientific computations or anything else that is compute-intensive and can utilize GPUs.
- ☐ Highly available  
Most demanding availability requirements. Both the master and the nodes are replicated across multiple zones.

**'Standard cluster' template**

Continuous integration, web serving, backends. Best choice for further customization or if you are not sure what to choose.

You will be billed for the 3 nodes (VM instances) in your cluster. [Learn more](#)

☐ Some fields can't be changed after the cluster is created. Hover over the help icons to learn more. [Dismiss](#)

Name

Location type ☒ Zonal  
☐ Regional

Zone

Master version

**Node pools**

Node pools are separate instance groups running Kubernetes in a cluster. You may add node pools in different zones for higher availability, or add node pools of different type machines. To add a node pool, click [Edit](#). [Learn more](#)

**default-pool**

Number of nodes

Machine type  3.75 GB memory [Customize](#)

[Upgrade your account](#) to create instances with up to 96 cores

Auto-upgrade: On  
[Advanced edit](#)

[+ Add node pool](#)

[Advanced options](#)

[Create](#) [Cancel](#) [Equivalent REST or command line](#)

Create a cluster

You will see many optional settings for creating a cluster in Create cluster page

Cluster template contains pre-configured clusters that will apply to create new cluster as soon as you select one.

For our case, we will select the first cluster option.

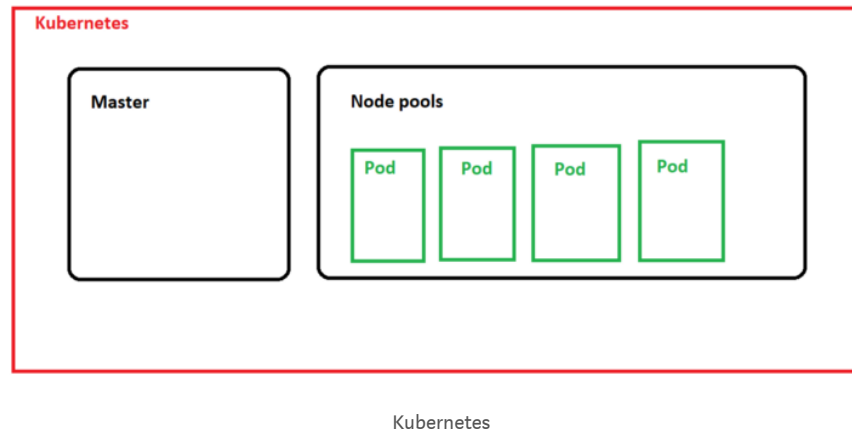
In Name, you can select any name

In location, you can select the cluster to be regional or zonal

In Zone, select the location near to your Users or application.

**In Master version, select the Kubernetes Master version.**

In Node pools, select the number of machines and machine types you require to make a cluster,



there are several other options in advanced features

**Load balancing**

☒ Enable HTTP load balancing ?

**Network security**

☐ Private cluster ?


☐ Enable master authorized networks ?

☐ Enable network policy ?

**Security**

☒ Enable basic authentication ?

☒ Issue a client certificate ?

 Starting with version 1.12, clusters will have basic authentication and client certificate issuance disabled by default. [Disable them now](#)

☐ Enable legacy authorization ?

☐ Enable binary authorization (beta) ?

**Metadata**

Description (Optional) ?

**Labels** (Optional)

To organize your project, add arbitrary labels as key/value pairs to your resources. Use labels to indicate different environments, services, teams, and so on. [Learn more](#)

+ Add label

**Additional features**

☒ Enable Stackdriver Logging service ?

☒ Enable Stackdriver Monitoring service ?

☐ Try the new Stackdriver beta Monitoring and Logging experience

The beta experience increases observability by aggregating incidents, system metrics, and logs into one single view

☐ Enable Cloud TPU (beta) ?

☐ Enable Kubernetes alpha features in this cluster ?

☐ Enable Kubernetes Dashboard ?

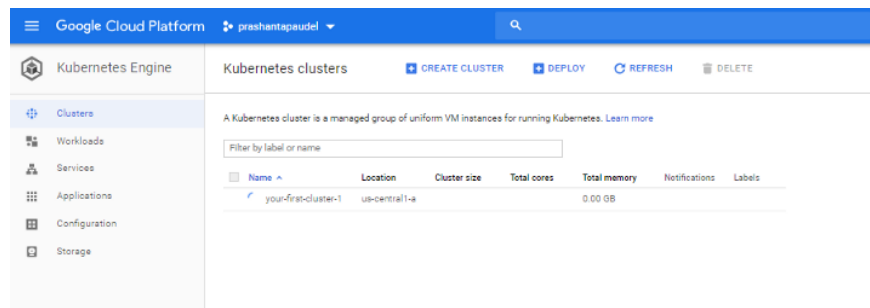
Create

Reset

Equivalent [REST](#) or [command line](#)

advanced features

Click Create and wait for a while until the cluster is built.

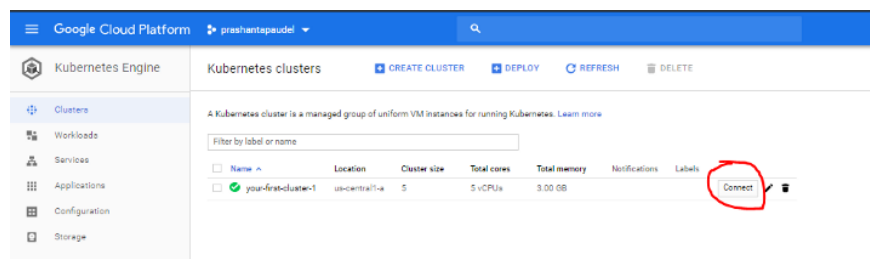


in process

the gcloud command for the same operation is

gcloud beta container—project “prashantapaudel-219410” clusters  
 create “your-first-cluster-1”—zone “us-central1-a”—username  
 “admin”—cluster-version “1.9.7-gke.6”—machine-type “f1-micro”—  
 image-type “COS”—disk-type “pd-standard”—disk-size “30”—scopes  
 “https://www.googleapis.com/auth/compute”, “https://www.googleap  
 is.com/auth/devstorage.read\_only”, “https://www.googleapis.com/aut  
 h/logging.write”, “https://www.googleapis.com/auth/monitoring”, “htt  
 ps://www.googleapis.com/auth/servicecontrol”, “https://www.google  
 apis.com/auth/service.management.readonly”, “https://www.googleap  
 is.com/auth/trace.append”—num-nodes “5”—enable-cloud-logging—  
 enable-cloud-monitoring—network “projects/prashantapaudel-  
 219410/global/networks/default”—subnetwork  
 “projects/prashantapaudel-219410/regions/us-  
 central1/subnetworks/default”—addons  
 HorizontalPodAutoscaling,HttpLoadBalancing,KubernetesDashboard  
 —enable-autoupgrade—enable-autorepair

After the cluster is ready you will see the connect button besides cluster  
 in Dashboard



Cluster page

Clicking the name of the cluster you can check the configuration of the cluster

✓ your-first-cluster-1

[Details](#) [Storage](#) [Nodes](#)

---

Cluster

Master version	1.9.7-gke.6	<a href="#">Upgrade available</a>
Endpoint	35.238.42.112	<a href="#">Show credentials</a>
Client certificate	Enabled	
Binary authorization	Disabled	
Kubernetes alpha features	Disabled	
Total size	5	
Master zone	us-central1-a	
Node zones	us-central1-a	
Network	<a href="#">default</a>	
Subnet	<a href="#">default</a>	
VPC-native (alias IP)	Disabled	
Pod address range	10.4.0.0/14	
Stackdriver Logging	Enabled	
Stackdriver Monitoring	Enabled	
Private cluster	Disabled	
Master authorized networks	Disabled	
Network policy	Disabled	
Legacy authorization	Disabled	
Maintenance window	Any time	
Cloud TPU	Disabled	

Labels

None

[Add-ons](#)

[Permissions](#)



your-first-cluster-1

Details Storage Nodes

Persistent volumes

Filter persistent volumes

No matching results

Storage classes

Filter storage classes

Name	Provisioner	Type	Zone
standard	kubernetes.io/gce-pd	pd-standard	

Nodes

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
gke-your-first-cluster-1-pool-1-d4a1b41d-m0	Ready	348 mCPU	940 mCPU	540.2 MiB	622.85 MiB	0 B	0 B
gke-your-first-cluster-1-pool-1-d4a1b41d-m1	Ready	350 mCPU	940 mCPU	440.4 MiB	622.85 MiB	0 B	0 B
gke-your-first-cluster-1-pool-1-d4a1b41d-m2	Ready	400 mCPU	940 mCPU	325.08 MiB	622.85 MiB	0 B	0 B
gke-your-first-cluster-1-pool-1-d4a1b41d-m3	Ready	233 mCPU	940 mCPU	371.2 MiB	622.85 MiB	0 B	0 B
gke-your-first-cluster-1-pool-1-d4a1b41d-m4	Ready	200 mCPU	940 mCPU	208.72 MiB	622.85 MiB	0 B	0 B

Nodes

In this way, we deployed the Kubernetes engine cluster with 5 nodes.

*In gcloud*

To do the same thing in gcloud use the command, remember there are so many scopes that need to be added for making it similar to above.

**This command just starts the cluster with default settings.**

```
$ gcloud container clusters create anothercluster --
region=us-central1-a
WARNING: Starting in 1.12, new clusters will have basic
authentication disabled by default. Basic authentication can
be enabled (or disabled) manually using the '--[no-]enable-
basic-auth' flag.
WARNING: Starting in 1.12, new clusters will not have a
client certificate issued. You can manually enable (or
disable) the issuance of the client certificate using the '--
[no-]issue-client-certificate'
```

```

flag.
WARNING: Currently VPC-native is not the default mode during
cluster creation. In the future, this will become the
default mode and can be disabled using `--no-enable-ip-
alias` flag. Use `[no-enable-ip-alias` flag to suppress
this warning.
WARNING: Starting in 1.12, default node pools in new
clusters will have their legacy Compute Engine instance
metadata endpoints disabled by default. To create a cluster
with legacy instance metadata endpoints disabled in the
default node pool, run `clusters create` with the flag `--
metadata disable-legacy-endpoints=true`.
This will enable the autorepair feature for nodes. Please
see https://cloud.google.com/kubernetes-engine/docs/node-
auto-repair for more information on node autorepairs.
WARNING: Starting in Kubernetes v1.10, new clusters will no
longer get compute-rw and storage-ro scopes added to what is
specified in --scopes (though the latter will remain
included in the default --scopes). To use these scopes, add
them explicitly to --scopes. To use the new behavior, set
container/new_scopes_behavior property (gcloud config set
container/new_scopes_behavior true).
Creating cluster anothercluster in us-central1-a...done.
Created
[https://container.googleapis.com/v1/projects/prashantapaude
l-219410/zones/us-central1-a/clusters/anothercluster].
To inspect the contents of your cluster, go to:
https://console.cloud.google.com/kubernetes/workload_/gcloud
/us-central1-a/anothercluster?project=prashantapaudel-219410
kubeconfig entry generated for anothercluster.
NAME             LOCATION     MASTER_VERSION  MASTER_IP
MACHINE_TYPE     NODE_VERSION  NUM_NODES      STATUS
anothercluster   us-central1-a  1.9.7-gke.6     35.238.5.216
n1-standard-1    1.9.7-gke.6   3              RUNNING
prashantagcppaudel@cloudshell:~ (prashantapaudel-219410)$

```

## Connect to the Cluster

To connect to the cluster use the command

```

$ gcloud container clusters get-credentials anothercluster -
--zone us-central1-a --project prashantapaudel-219410
Fetching cluster endpoint and auth data.
kubeconfig entry generated for anothercluster.

```

List the clusters

```

prashantagcppaudel@cloudshell:~ (prashantapaudel-219410)$
gcloud container clusters list

```

```

NAME                LOCATION      MASTER_VERSION
MASTER_IP          MACHINE_TYPE  NODE_VERSION  NUM_NODES
STATUS
anothercluster      us-central1-a 1.9.7-gke.6
35.238.5.216        n1-standard-1 1.9.7-gke.6   3
RUNNING
your-first-cluster-1 us-central1-a 1.9.7-gke.6
35.238.42.112       f1-micro      1.9.7-gke.6   5
RUNNING
prashantagcpaudel@cloudshell:~ (prashantapaudel-219410)$

```

## Deploying a container application to Kubernetes Engine using pods

### Step 1: Build the container image

First of all, we need to have an application and docker file ready.

We will use already available hello app from Github.

First, download the app

```

$ git clone
https://github.com/GoogleCloudPlatform/kubernetes-engine-
samples
Cloning into 'kubernetes-engine-samples'...
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 476 (delta 8), reused 8 (delta 4), pack-reused
447
Receiving objects: 100% (476/476), 389.76 KiB | 0 bytes/s,
done.
Resolving deltas: 100% (201/201), done.

```

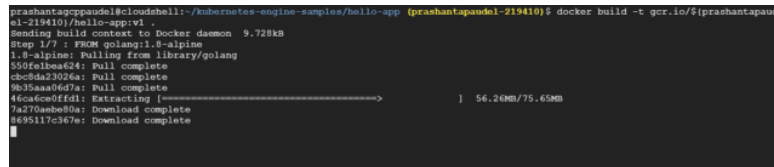
Set the `PROJECT_ID` environment variable in your shell by retrieving the pre-configured project ID on `gcloud` by running the command below:

```
$ export PROJECT_ID="$(gcloud config get-value project -q)"
Your active configuration is: [cloudshell-7515]
```

The value of `PROJECT_ID` will be used to tag the container image for pushing it to your private Container Registry.

**To build the container image** of this application and tag it for uploading, run the following command:

```
docker build -t gcr.io/${PROJECT_ID}/hello-app:v1
```



```
prashantapandel@cloudshell:~/kubernetes-engine-samples/hello-app (prashantapandel-219410)$ docker build -t gcr.io/prashantapandel-219410/hello-app:v1
Sending build context to Docker daemon  9.728kB
Step 1/7 : FROM golang:1.8-alpine
1.8-alpine: Pulling from library/golang
550fe1ba624: Pull complete
cb8da23026a: Pull complete
0b3baa0d67a: Pull complete
46ca6ce0ffdl: Extracting [=====] 56.24MB/75.65MB
7a270a8be80a: Download complete
869517c367e: Download complete
```

building docker

This command instructs Docker to build the image using the `Dockerfile` in the current directory and tag it with a name, such as `gcr.io/my-project/hello-app:v1`. The `gcr.io` prefix refers to Google Container Registry, where the image will be hosted.

Running this command does not upload the image yet.

Run `docker images` command to verify that the build was successful:

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID
gcr.io/219410/hello-app	v1	367da9716bc1
4 minutes ago	10.3MB	
<none>	<none>	be6ee6935fed
4 minutes ago	263MB	
alpine	latest	196d12cf6ab1
6 weeks ago	4.41MB	
golang	1.8-alpine	4cb86d3661bf
8 months ago	257MB	

## Step 2: Upload the container image

First, run the auth command

```
gcloud auth configure-docker

prashantagcppaudel@cloudshell:~/kubernetes-engine-
samples/hello-app (prashantapaudel-219410)$ gcloud auth
configure-docker
WARNING: Your config file at
[/home/prashantagcppaudel/.docker/config.json] contains
these credential helper entries:

{
  "credHelpers": {
    "gcr.io": "gcr",
    "us.gcr.io": "gcr",
    "asia.gcr.io": "gcr",
    "staging-k8s.gcr.io": "gcr",
    "eu.gcr.io": "gcr"
  }
}
These will be overwritten.
The following settings will be added to your Docker config
file
located at [/home/prashantagcppaudel/.docker/config.json]:
{
  "credHelpers": {
    "gcr.io": "gcloud",
    "us.gcr.io": "gcloud",
    "eu.gcr.io": "gcloud",
    "asia.gcr.io": "gcloud",
    "staging-k8s.gcr.io": "gcloud",
    "marketplace.gcr.io": "gcloud"
  }
}

Do you want to continue (Y/n)? y

Docker configuration file updated.
prashantagcppaudel@cloudshell:~/kubernetes-engine-
samples/hello-app (prashantapaudel-219410)$
```

You can now use the Docker command-line tool to upload the image to your Container Registry:

```
docker push gcr.io/${PROJECT_ID}/hello-app:v1
```

## Step 3: Run your container locally (optional)

To test your container image using your local Docker engine, run the following command:

```
docker run --rm -p 8080:8080 gcr.io/${PROJECT_ID}/hello-app:v1
```

If you're on Cloud Shell, you can click "Web preview" button on the top right to see your application running in a browser tab. Otherwise, open a new terminal window (or a Cloud Shell tab) and run to verify if the container works and responds to requests with "Hello, World!":

## Step 4: Create a container cluster

Now that the container image is stored in a registry, you need to create a container cluster to run the container image. A cluster consists of a pool of Compute Engine VM instances running Kubernetes, the open source cluster orchestration system that powers GKE.

Once you have created a GKE cluster, you use Kubernetes to deploy applications to the cluster and manage the applications' lifecycle.

Run the following command to create a three-node cluster named

```
hello-cluster :
```

```
gcloud container clusters create hello-cluster --num-nodes=3
```

It may take several minutes for the cluster to be created. Once the command has completed, run the following command and see the cluster's three worker VM instances:

```
gcloud compute instances list
```

which will show all compute instances including containers

```
prashantagcpaudel@cloudshell:~/kubernetes-engine-
samples/hello-app (prashantapaudel-219410)$ gcloud compute
instances list
NAME                                                    ZONE
MACHINE_TYPE    PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP
STATUS
gke-anothercluster-default-pool-b5fd2208-9xxf  us-central1-a
n1-standard-1                10.128.0.7   104.197.136.126
RUNNING
gke-anothercluster-default-pool-b5fd2208-jx3k  us-central1-a
n1-standard-1                10.128.0.9   35.225.236.118
RUNNING
gke-anothercluster-default-pool-b5fd2208-kbsm  us-central1-a
n1-standard-1                10.128.0.8   35.192.54.14
RUNNING
gke-your-first-cluster-1-pool-1-da4e9b41-0m4c  us-central1-a
f1-micro                  10.128.0.3   104.154.106.120
RUNNING
gke-your-first-cluster-1-pool-1-da4e9b41-5pks  us-central1-a
f1-micro                  10.128.0.6   35.193.105.242
RUNNING
gke-your-first-cluster-1-pool-1-da4e9b41-dk81  us-central1-a
f1-micro                  10.128.0.2   35.239.24.9
RUNNING
gke-your-first-cluster-1-pool-1-da4e9b41-hfz1  us-central1-a
f1-micro                  10.128.0.4   35.238.147.20
RUNNING
gke-your-first-cluster-1-pool-1-da4e9b41-knlg  us-central1-a
f1-micro                  10.128.0.5   35.184.0.99
RUNNING
prashantagcpaudel@cloudshell:~/kubernetes-engine-
samples/hello-app (prashantapaudel-219410)$
```

## Step 5: Deploy your application

To deploy and manage applications on a GKE cluster, you must communicate with the Kubernetes cluster management system. You typically do this by using the `kubectl` command-line tool.

Kubernetes represents applications as Pods, which are units that represent a container (or group of tightly-coupled containers). The Pod is the smallest deployable unit in Kubernetes. In this tutorial, each Pod contains only your `hello-app` container.

The `kubectl run` command below causes Kubernetes to create a Deployment named `hello-web` on your cluster. The Deployment

manages multiple copies of your application, called replicas, and schedules them to run on the individual nodes in your cluster. In this case, the Deployment will be running only one Pod of your application.

Run the following command to deploy your application, listening on port 8080:

```
kubectl run hello-web --image=gcr.io/${PROJECT_ID}/hello-app:v1 --port 8080
```

To see the Pod created by the Deployment, run the following command:

NAME	READY	STATUS	RESTARTS
AGE			
hello-web-4017757401-px7tx	1/1	Running	0
3s			

## Step 6: Expose your application to the Internet

By default, the containers you run on GKE are not accessible from the Internet, because they do not have external IP addresses. You must explicitly expose your application to traffic from the Internet, run the following command:

```
kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-port 8080
```

## Step 7: Scale up your application

You add more replicas to your application's Deployment resource by using the `kubectl scale` command. To add two additional replicas to your Deployment (for a total of three), run the following command:



```
kubectl scale deployment hello-web --replicas=3
```

You can see the new replicas running on your cluster by running the following commands:

```
kubectl get deployment hello-web
```

Output:

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
hello-web	3	3	3	2	1m

```
kubectl get pods
```

Output:

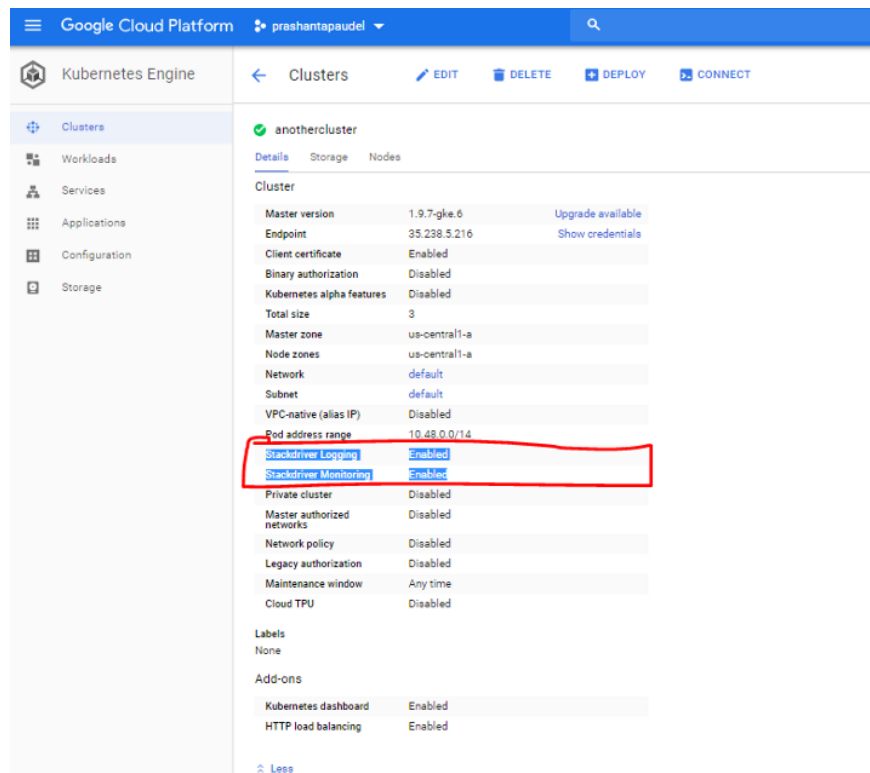
NAME	READY	STATUS	RESTARTS
hello-web-4017757401-ntgdb	1/1	Running	0
hello-web-4017757401-pc4j9	1/1	Running	0
hello-web-4017757401-px7tx	1/1	Running	0

Now, you have multiple instances of your application running independently of each other and you can use the `kubectl scale` command to adjust the capacity of your application.

The load balancer you provisioned in the previous step will start routing traffic to these new replicas automatically.

# Configuring Kubernetes Engine application monitoring and logging

Now it is very easy to start cluster monitoring by just selecting monitoring while creating the cluster



stackdriver monitoring

## Installing Kubernetes Monitoring

### Beta

This is a Beta release of **Stackdriver Kubernetes Monitoring**. This feature is not covered by any SLA or deprecation policy and might be subject to backward-incompatible changes.

The page explains how to install the Beta release of Stackdriver Kubernetes Monitoring. This is an opt-in release that replaces the legacy Stackdriver support described in the GKE guides, Monitoring, and Logging. This Beta release is intended only for Google Kubernetes Engine (GKE).

You can install this new release when you create a new GKE cluster, or you can upgrade an existing GKE cluster. If you do not opt into this release, then your new clusters use the present Stackdriver support, also called the **GA legacy support**.

## Before you begin

**Caution:** If you intend to upgrade an existing cluster, read the information about incompatibilities.

- You must be an **Owner** of the project containing your cluster. The project must be monitored by a Workspace.
- You will be required to use Kubernetes version 1.10.6 or 1.11.2 (or later, if available).

## Creating a new cluster

You must opt into this Beta release when you create a new cluster using the GKE console or the Cloud SDK's `gcloud` command-line tool.

## CONSOLE

1. Go to the GKE **Kubernetes Clusters** page for your project. The following button takes you there:
2. Select **Create Cluster** at the top of the page.
3. In the dialog, specify the following fields in addition to whichever other properties you want in your cluster. For more information, see [Creating a Cluster](#).
  - **Cluster Version:** 1.10.6 or later; or 1.11.2 or later
  - **Try the new Beta Monitoring and Logging experience:** CHECKED
  - If you do not see this option, then you have not selected a cluster version of 1.10.6 or 1.11.2 (or later, if available). If you see the option but it is grayed out, then be sure that both the **Stackdriver Logging** and **Stackdriver Monitoring** options are `Enabled` (the default).

- This part of the options panel should look like the following:

**Legacy Authorization** ?

Disabled

**Stackdriver Logging** ?

Enabled

**Stackdriver Monitoring** ?

Enabled

☒ Try the new Beta Monitoring and Logging experience

The beta experience increases observability by aggregating incidents, system metrics, and logs into one single view

1. Click **Create** to create the cluster. This may take some time.

## Upgrading an existing cluster

The following table tells you how to upgrade your cluster to the managed Beta release of Stackdriver Kubernetes Monitoring. There are differences depending on what previous Stackdriver support the cluster is running, which you can tell by clicking your cluster's name in the GKE console.

Current Stackdriver support listed Stackdriver status  
in console how to upgradeBeta managed releaseEnabled v2(beta)You  
only need to upgrade your Kubernetes version. Stackdriver is updated  
when Kubernetes is.GA legacyEnabledUse the following Upgrade  
Instructions.noneDisabledUse the following Upgrade Instructions.non-  
managed,  
manual Beta1Disabled and the cluster contains a `stackdriver-agents`  
namespaceDelete the current agents:

```
kubectl delete ns stackdriver-agents
```

Then use the following Upgrade Instructions.

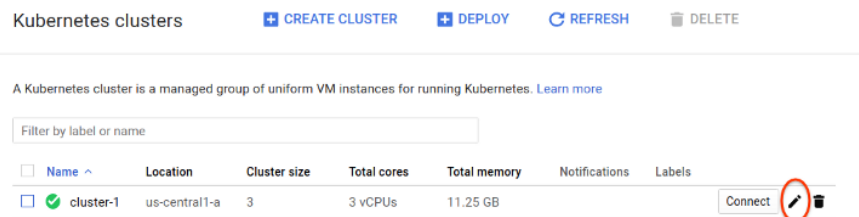
Notes:

- 1 Presently, this can only happen if you manually installed updated agents. Manual installation is no longer needed or recommended.

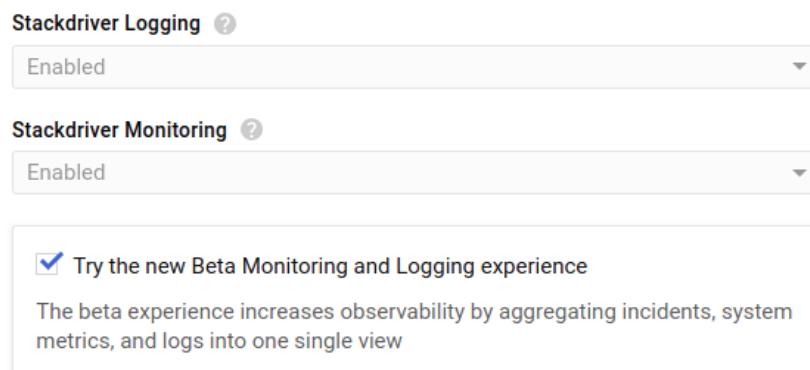
The following instructions upgrade your GKE cluster to the managed Beta release of Stackdriver Kubernetes Monitoring. Check the previous table that mentions some corner cases.

## CONSOLE

1. Go to the GKE **Kubernetes Clusters** page for your project. The following button takes you there:
2. Upgrade your cluster to Kubernetes version 1.10.6 or 1.11.2 (or later, if available). For instructions, see [Upgrading Clusters](#). Allow the upgrade to complete.
3. Click the **Edit** icon (edit) for your cluster:



1. Scroll down to the Stackdriver section and check the box **Try the new Beta Monitoring and Logging experience**. You might have to **enable** the **Stackdriver Monitoring** and **Logging** choices also:



1. If you have not upgraded to the required Kubernetes version, you will not see the **Try the new Beta...** option.
2. Click **Save** at the bottom of the page.

**Next step:** Go to Checking your installation on this page.

## Checking your installation

To check that Stackdriver Monitoring is running properly, wait a few minutes and then examine your cluster. Use either the console or

`gcloud` to see that the Stackdriver pods are running:

## CONSOLE

1. From the GCP console, go to **Stackdriver > Monitoring**:
2. In Monitoring, go to **Resources > Kubernetes BETA** for the Workspace containing your GCP project.
3. You can see all the clusters in your Workspace that are using Stackdriver Kubernetes Monitoring.
4. Select the **Workload** tab.
5. Expand your cluster and the `kube-system` namespace. You should see running pods in the `kube-system` namespace with the following names:
  - `fluentd-gcp-...` : the Stackdriver Logging agent.
  - `heapster-...` : the Monitoring agent.
  - `metadata-...` : the Stackdriver metadata agent.

