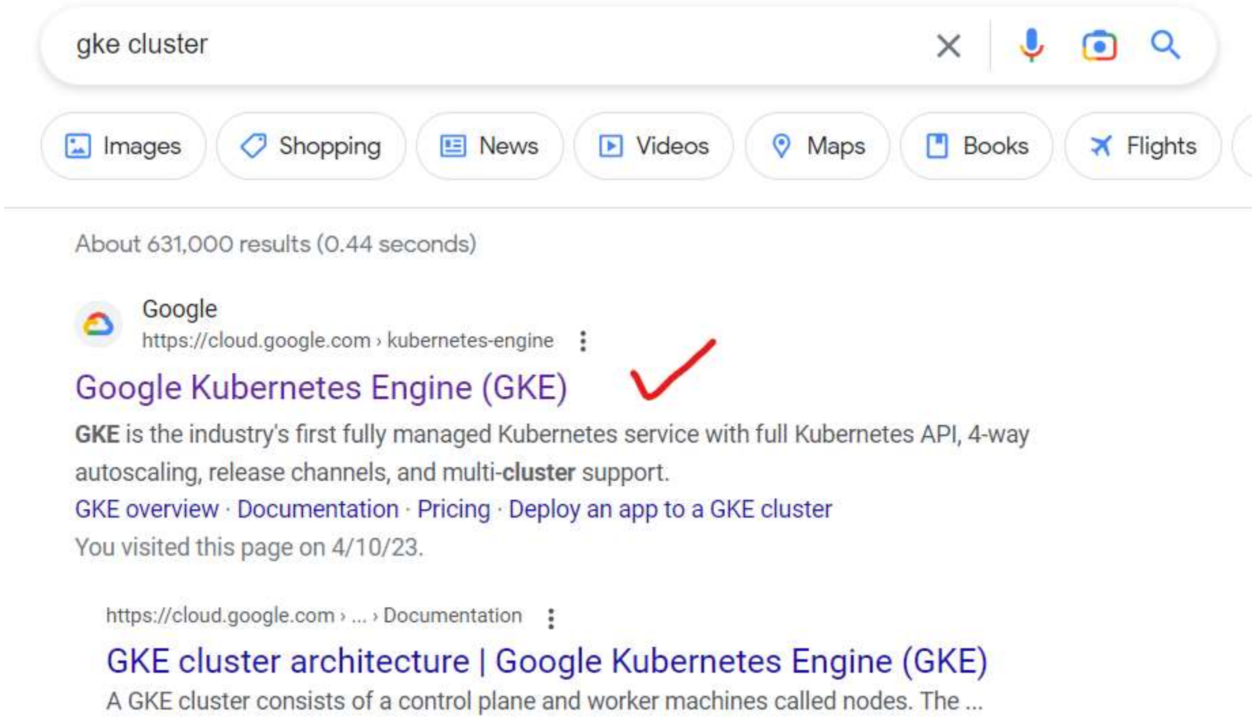
AmazingDevOps

Google Kubernetes Engine (GKE)

Here are the steps to create an account for Google Kubernetes Engine (GKE):

1. Go to the Google Cloud Console at https://console.cloud.google.com/.



gke cluster

About 631,000 results (0.44 seconds)

Google
https://cloud.google.com › kubernetes-engine

Google Kubernetes Engine (GKE)

GKE is the industry's first fully managed Kubernetes service with full Kubernetes API, 4-way autoscaling, release channels, and multi-**cluster** support.

GKE overview · Documentation · Pricing · Deploy an app to a GKE cluster

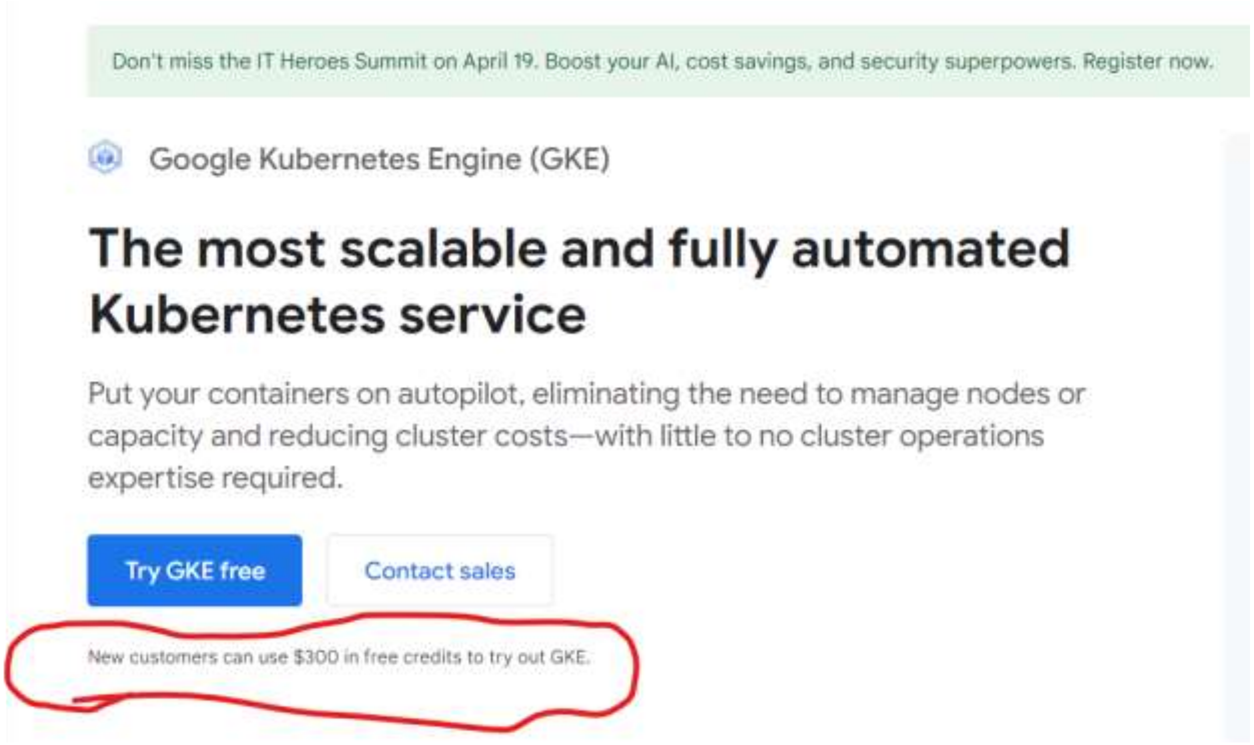
You visited this page on 4/10/23.

https://cloud.google.com › ... › Documentation

GKE cluster architecture | Google Kubernetes Engine (GKE)

A GKE cluster consists of a control plane and worker machines called nodes. The ...

2. If you already have a Google account, sign in. Otherwise, click the "Create an account" button and follow the instructions to create a new Google account.


Don't miss the IT Heroes Summit on April 19. Boost your AI, cost savings, and security superpowers. Register now.

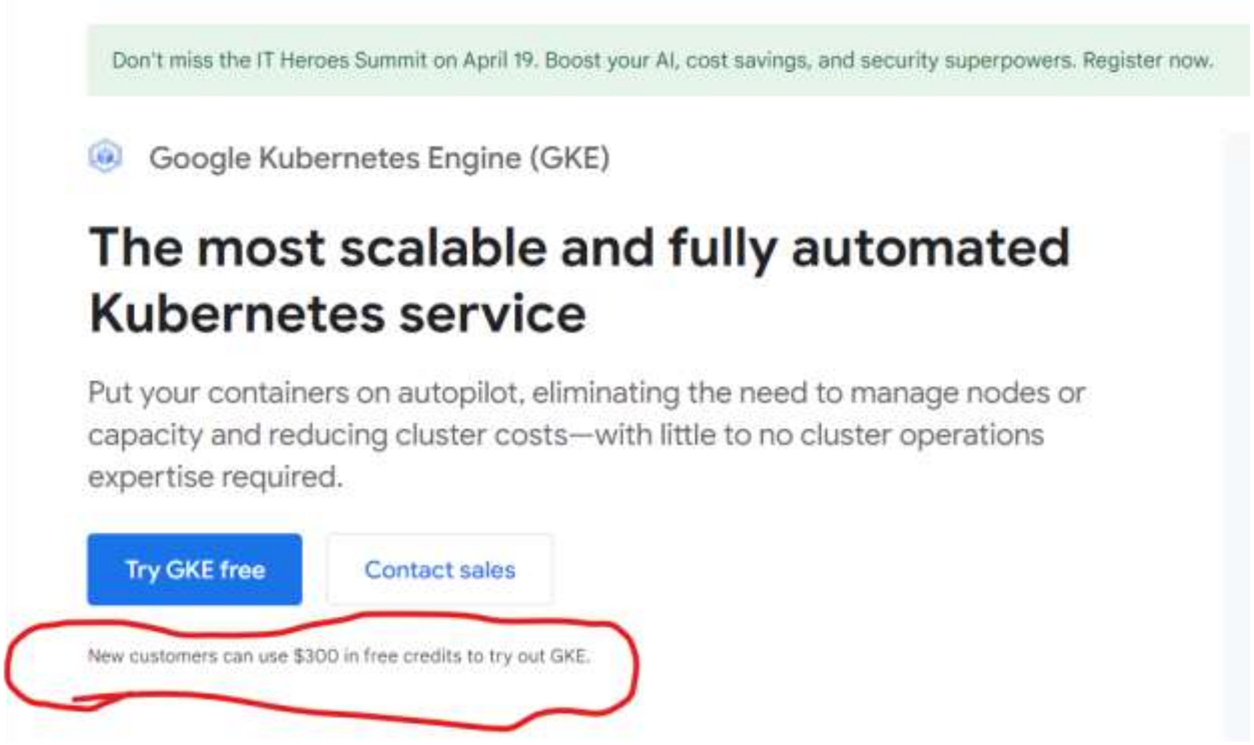
Google Kubernetes Engine (GKE)

The most scalable and fully automated Kubernetes service

Put your containers on autopilot, eliminating the need to manage nodes or capacity and reducing cluster costs—with little to no cluster operations expertise required.

New customers can use $300 in free credits to try out GKE.

3. Once you are signed in, navigate to the Google Cloud Console dashboard.


Don't miss the IT Heroes Summit on April 19. Boost your AI, cost savings, and security superpowers. Register now.

Google Kubernetes Engine (GKE)

The most scalable and fully automated Kubernetes service

Put your containers on autopilot, eliminating the need to manage nodes or capacity and reducing cluster costs—with little to no cluster operations expertise required.

New customers can use $300 in free credits to try out GKE.

4. Click on Try GKE free and provide all the information needed to get a credit of $ 300.
5. ENABLE Kubernetes Engine API
6. Create a cluster. This may take 5-10 minutes.
7. Connect to the cluster.
8. Open a new terminal and run the following three commands.

Here are 20 common kubectl commands frequently used by DevOps engineers while working with GKE clusters:

To create a new Kubernetes namespace using the **kubectl** command-line tool, you can use the following command:

Kubernetes namespace

In Kubernetes, a namespace is a virtual cluster that provides a way to partition resources and create logical boundaries within a Kubernetes cluster. **Namespaces are commonly used to separate workloads and provide multi-tenancy within a cluster.**

Each Kubernetes **namespace provides a separate namespace scope for its resources, including pods, services, replication controllers, and others.** This means that resources created in one namespace are isolated from resources created in other namespaces, even if they have the same name.

By using namespaces, you can organize your resources and workloads based on different criteria, such as teams, applications, or environments. This can help you to avoid naming conflicts and provide better isolation and security for your workloads.

Kubernetes also includes a default namespace, which is used if no other namespace is specified when creating a resource. You can create additional namespaces using the **kubectl create namespace <namespace-name>** command, and switch between namespaces using the **kubectl config set-context --current --namespace=<namespace-name>** command.

Overall, namespaces provide a way to logically partition resources within a Kubernetes cluster, and can help to improve resource utilization, security, and management of your workloads.

# kubectl create namespace <namespace-name>

Replace **<namespace-name>** with the desired name of your new namespace.

For example, to create a namespace named "my-namespace", you can run the following command:

Kubernetes Namespace to a Dummy

Imagine you have a big toy box with lots of different toys in it, such as building blocks, toy cars, and stuffed animals. It can be hard to find the toy you want in the toy box, especially if there are lots of toys mixed.

In a similar way, a Kubernetes cluster can have lots of different applications and services running in it. It can be hard to manage and organize all these resources if they're all mixed.

This is where namespaces come in. You can think of a namespace as a special box that you can put some of your toys in. For example, you could put all your building blocks in one namespace, and all your toy cars in another namespace.

By separating your toys into different namespaces, you can find and manage them more easily. You can also share the toy box with other people, and each person can have their own namespace to put their toys in.

In Kubernetes, namespaces work in a similar way. You can create different namespaces to separate your applications and services based on different criteria, such as teams, projects, or environments. Each namespace has its own set of resources, like pods and services, which are separate from the resources in other namespaces.

Overall, namespaces provide a way to organize and manage your resources in Kubernetes, and make it easier to find and manage your workloads.

```
# kubectl create namespace my-namespace
```

After running the command, you can use the following command to confirm that the namespace was created:

```
# kubectl get namespaces
```

This will display a list of all the namespaces in your cluster, including the new one you just created.

To delete a Kubernetes namespace using the **kubectl** command-line tool, you can use the following command:

# kubectl delete namespace <namespace-name>

Replace **<namespace-name>** with the name of the namespace you want to delete.

For example, to delete a namespace named "my-namespace", you can run the following command:

# kubectl delete namespace my-namespace

After running the command, you can use the following command to confirm that the namespace was deleted:

# kubectl get namespaces

This will display a list of all the remaining namespaces in your cluster, excluding the one you just deleted.

In Kubernetes, a node is a physical or virtual machine that is part of a Kubernetes cluster and is responsible for running containerized applications. A node is sometimes referred to as a worker node.

Kubernetes Nodes

Nodes are responsible for running containers, which are deployed as part of Kubernetes Pods. Each node has a container runtime (such as Docker or containerd) that is responsible for starting and stopping containers as needed.

Nodes also have a Kubelet process running on them, which is responsible for communicating with the Kubernetes control plane and managing the containers on the node. Additionally, nodes may have other Kubernetes components running on them, such as the kube-proxy, which is responsible for managing network connectivity between Pods on the node and other parts of the cluster.

Nodes can be added or removed from a Kubernetes cluster as needed, and workloads can be automatically scheduled across the available nodes using Kubernetes scheduling mechanisms. This allows applications to scale and run across multiple nodes for increased availability and reliability.

kubectl get nodes: Lists all the nodes in the cluster and their current status.

To check the nodes in a Kubernetes cluster using the kubectl command-line tool, you can use the following command:

# kubectl get nodes

This command will display a list of all the nodes in the cluster, including their name, status, age, and other information.

If you want to see more detailed information about the nodes, you can use the following command:

# kubectl describe nodes

This command will display detailed information about each node, including its resource usage, network settings, and other configuration details.

You can also use labels to filter the list of nodes. For example, if you have labeled your nodes with a specific key-value pair, you can use the following command to filter the list:

# kubectl get nodes -l <label-selector>

Replace <label-selector> with the label selector you want to use. For example, if you have labeled your nodes with app=web, you can use the following command to display a list of nodes with that label:

# kubectl get nodes -l app=web

Kubernetes pods

Imagine you have a suitcase, and you need to carry a few items in it, such as your clothes, shoes, and some toiletries. Now, instead of carrying these items in separate bags, you put them all together in your suitcase, so it's easier to manage and carry around.

In a similar way, a Kubernetes pod is like a suitcase that can hold one or more containers. The containers are like the items you put in the suitcase, which are grouped together for easier management.

For example, let's say you have an application that requires a web server and a database server. You could create a pod that includes both of these containers, so they can work together seamlessly. The pod would provide a common network and storage environment for the containers to communicate with each other.

Just like you can carry your suitcase around with you, Kubernetes can schedule and run your pod on a specific node in the cluster. If that node becomes unavailable or the pod needs to be scaled up to handle more traffic, Kubernetes can automatically move the pod to a different node or create additional replicas of the pod to handle the increased load.

Overall, pods make it easier to manage and deploy containerized applications in Kubernetes, by providing a way to group containers together and manage their lifecycle as a single unit.

# kubectl get pods: Lists all the pods running in the cluster.