The kubectl command is the command line utility used to interact with and manage the resources inside Kubernetes clusters. In this module you will learn about the kubectl command, how to connect it to Google Kubernetes Engine clusters, and use it to create, inspect, interact and delete Pods and other objects within Kubernetes clusters.

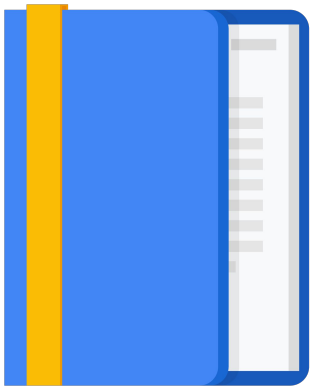
Learn how to ...

Work with the kubectl command.

Inspect the cluster and Pods.

View a Pod's console output.

Sign in to a Pod interactively.



Google Cloud

In this module, you'll learn how to work with the kubectl command to connect to GKE clusters, inspect the cluster and Pods, view a Pod's console output, and sign in interactively to a Pod.

Agenda

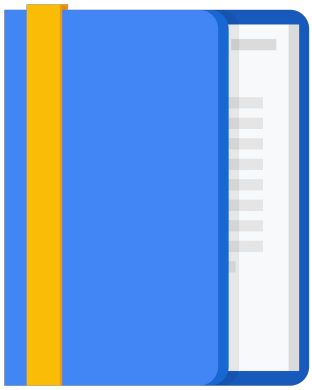
The kubectl command

Introspection

Lab: Deploying Google Kubernetes Engine Clusters from Cloud Shell

Lab: Upgrading Google Kubernetes Engine Clusters

Summary

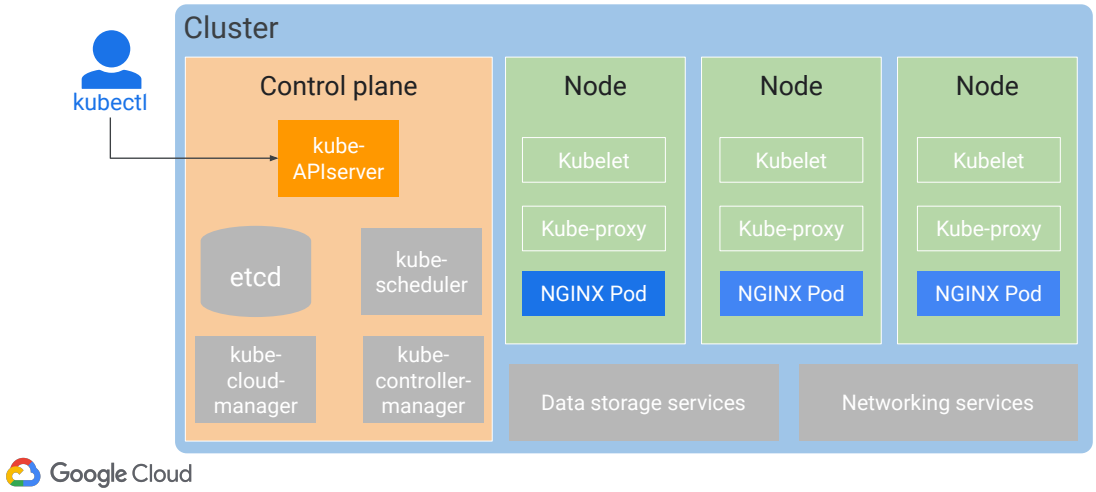


Google Cloud

Let’s start by discussing the kubectl command.

Kubectl is a utility used by administrators to control Kubernetes clusters. You use it to communicate with the Kube API server on your control plane.

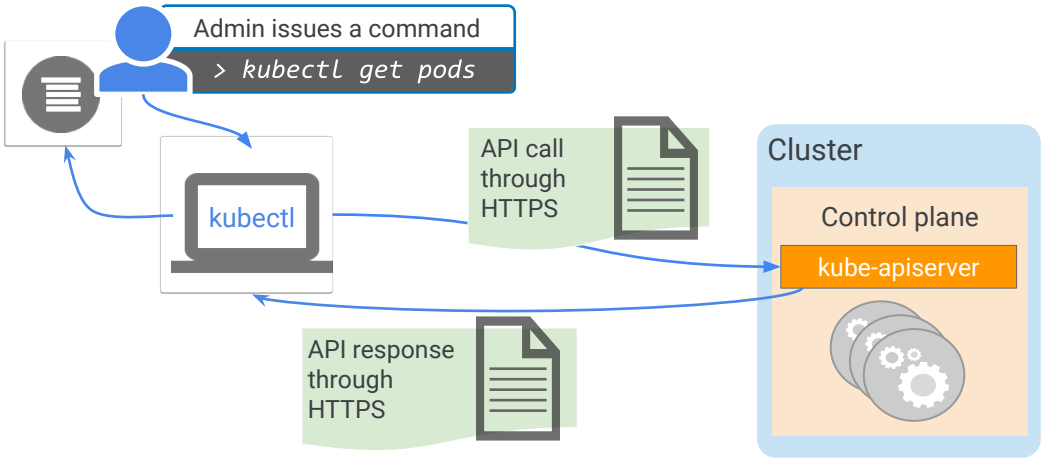
Kubectl transforms your command-line entries into API calls



Google Cloud

Kubectl transforms your command-line entries into API calls that it sends to the Kube API server within your selected Kubernetes cluster. Before it can do any work for you, kubectl must be configured with the location and credentials of a Kubernetes cluster.

Use kubectl to see a list of Pods in a cluster



Google Cloud

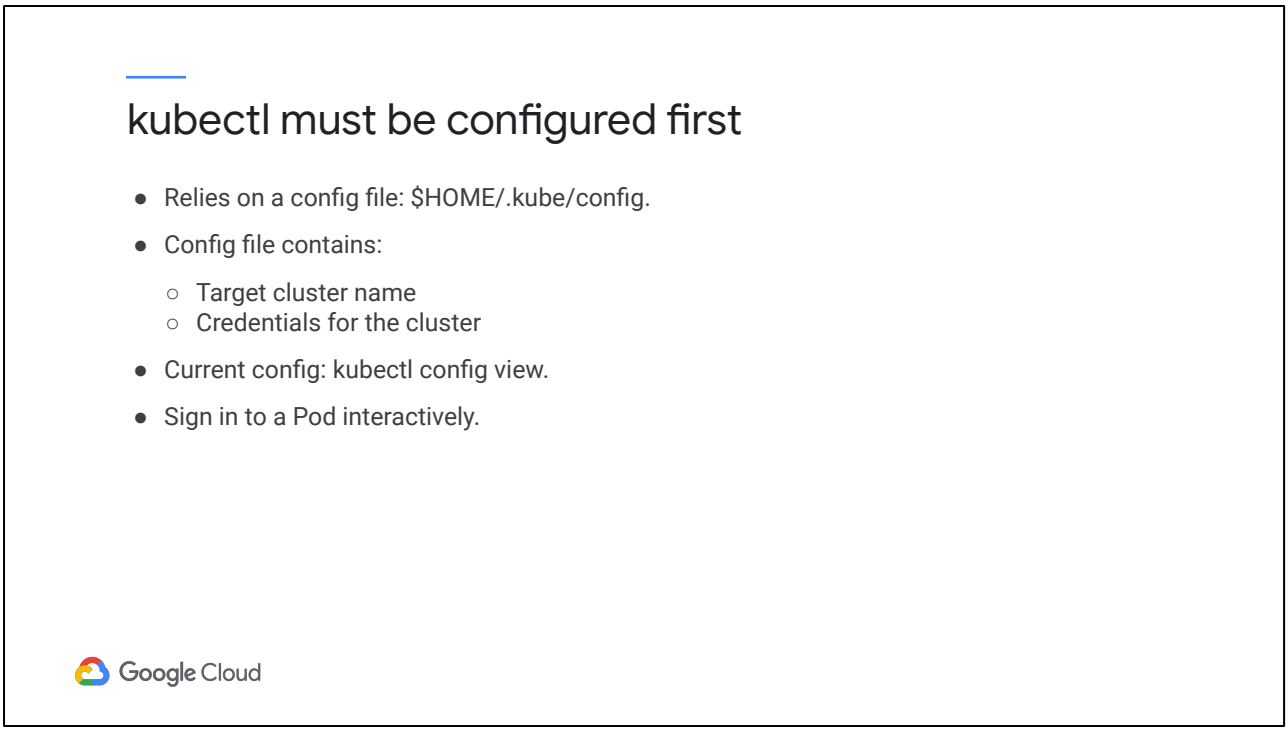
For example, take an administrator who wants to see a list of Pods in a cluster. After connecting kubectl to the cluster with proper credentials, the administrator can issue the kubectl ‘get pods’ command.

Kubectl converts this into an API call, which it sends to the Kube API server through HTTPS on the cluster’s control plane server.

The Kube API server processes the request by querying etcd.

The Kube API server then returns the results to kubectl through HTTPS.

Finally, kubectl interprets the API response and displays the results to the administrator at the command prompt.



Before you can use kubectl to configure your cluster, you must configure it first.

Kubectl stores its configuration in a file in your home directory in a hidden folder named .kube.

The configuration file contains the list of clusters and the credentials that you'll use to attach to each of those clusters. You may be wondering where you get these credentials. For GKE, the service provides them to you through the gcloud command. I'll show you how that works in a moment.

To view the configuration, you can either open the config file or use the kubectl command: 'config view'. Just to be clear here: kubectl config view tells you about the configuration of the kubectl *command itself*. Other kubectl commands tell you about the configurations of your cluster and workloads.

Connecting to a Google Kubernetes Engine cluster

```
$ gcloud container clusters \
  get-credentials [CLUSTER_NAME] \
  --zone [ZONE_NAME]
```

Google Cloud

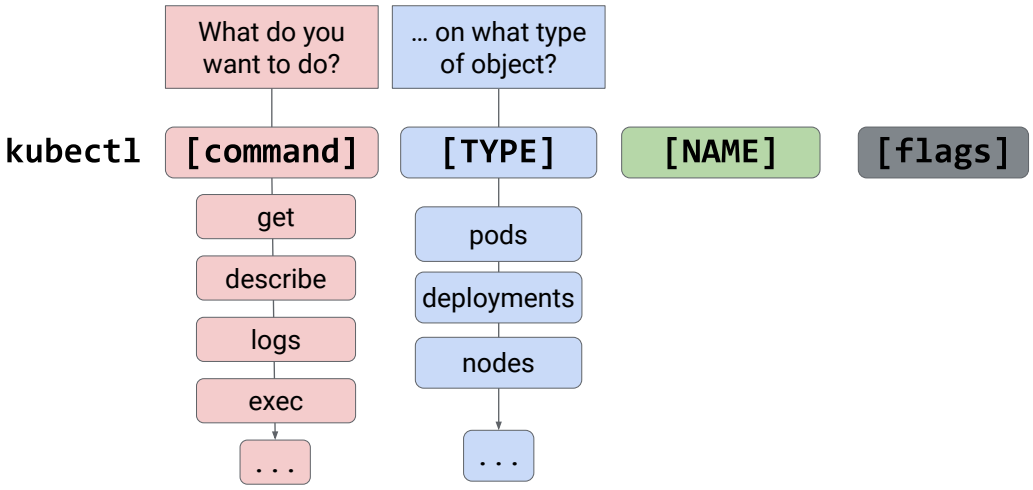
To connect to a GKE cluster with kubectl, retrieve your credentials for the specified cluster first. To do this, use the 'get-credentials' gcloud command in any other environment where you've installed the gcloud command-line tool and kubectl. Both of these tools are installed by default in the Cloud Shell.

The gcloud 'get-credentials' command writes configuration information into a config file in the .kube directory in the $HOME directory by default. If you rerun this command for a different cluster it'll update the config file with the credentials for the new cluster. You only need to perform this configuration process once per cluster in your Cloud Shell, because the .kube directory and its contents stay in your $HOME directory.

Can you figure out why the command is "gcloud get-credentials" rather than "kubectl get-credentials"? It's because the kubectl command requires credentials to work at all. The gcloud command is how authorized users interact with Google Cloud from the command line. The gcloud get-credentials command gives you the credentials you need to connect with a GKE cluster if you are authorized to do so.

In general, kubectl is a tool for administering the internal state of an existing cluster. But kubectl can't create new clusters or change the shape of existing clusters; for that you need the GKE control plane, which the gcloud command and the Cloud Console are your interfaces to.

The kubectl command syntax has several parts



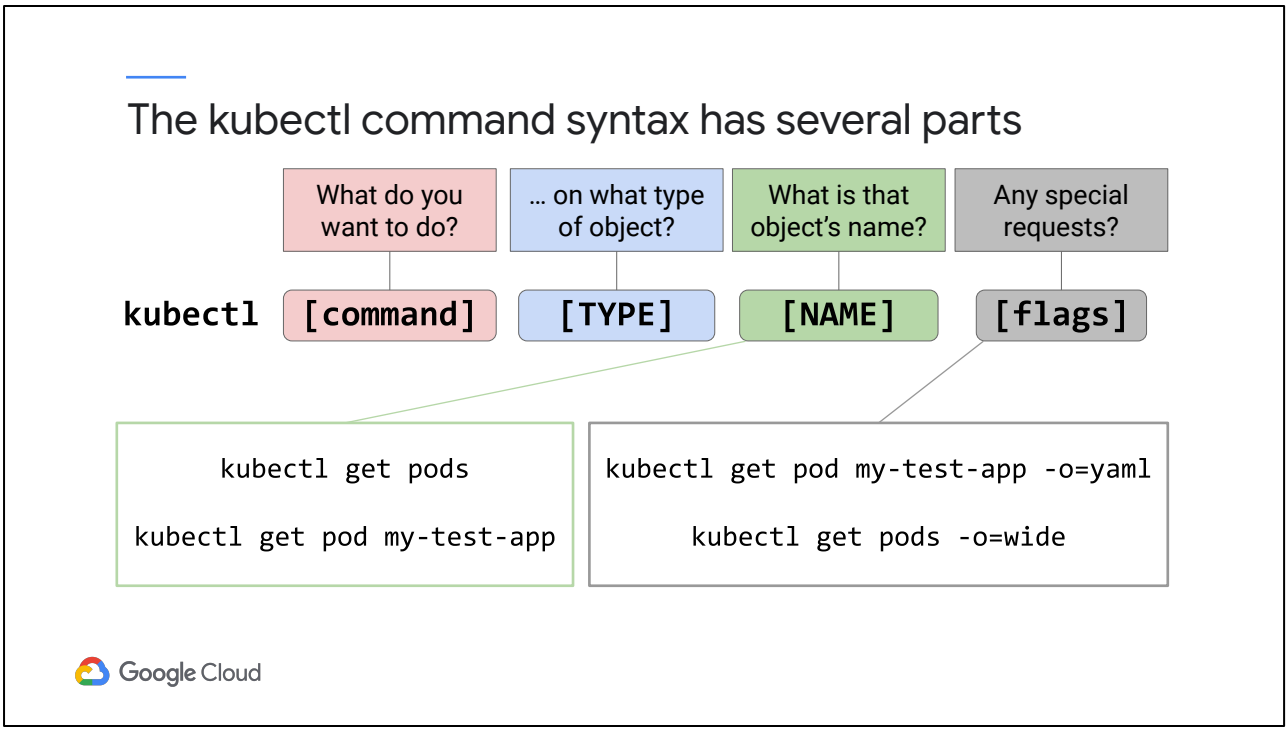
Google Cloud

Once the config file in the .kube folder has been configured, the kubectl command automatically references this file and connects to the default cluster without prompting you for credentials. Now let's talk about how to use the kubectl command. Its syntax is composed of several parts: the command, the type, the name, and optional flags.

'Command' specifies the action that you want to perform, such as get, describe, logs, or exec. Some commands show you information, while others allow you to change the cluster's configuration.

'TYPE' defines the Kubernetes object that the 'command' acts upon. For example, you could specify Pods, Deployments, nodes, or other objects, including the cluster itself.

TYPE used in combination with 'command' tells kubectl what you want to do and the type of object you want to perform that action on.



'NAME' specifies the object defined in 'TYPE.' The Name field isn't always needed, especially when you're using commands that list or show you information.
For example, if you run the command "kubectl get pods" without specifying a name, the command returns the list of all Pods. To filter this list you specify a Pod's name, such as "kubectl get pod my-test-app ". kubectl then returns information only on the Pod named 'my-test-app'.

Some commands support additional optional flags that you can include at the end of the command.
Think of this as making a special request, like formatting the output in a certain way. You could view the state of a Pod by using the command "kubectl get pod my-test-app -o=yaml". By the way, telling kubectl to give you output in YAML format is a really useful tool. You'll often want to capture the existing state of a Kubernetes object in a YAML file so that, for example, you can recreate it in a different cluster.

You can also use flags to display more information than you normally see. For instance, you can run the command "kubectl get pods -o=wide" to display the list of Pods in "wide" format, which means you see additional columns of data for each of the Pods in the list. One noteworthy piece of extra information you get in wide format: which Node each Pod is running on.

The kubectl command has many uses

- Create Kubernetes objects
- View objects
- Delete objects
- View and export configurations

Google Cloud

You can do many things with the kubectl command, from creating Kubernetes objects, to viewing them, deleting them, and viewing or exporting configuration files.

Just remember to configure kubectl first or to use the --kubeconfig or --context parameters, so that the commands you type are performed on the cluster you intended.

Agenda

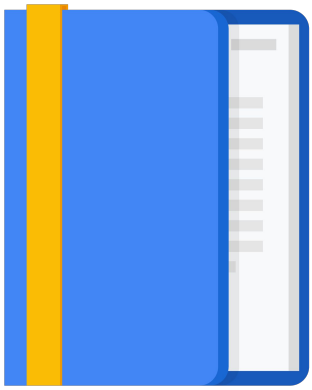
The kubectl command

Introspection

Lab: Deploying Google Kubernetes Engine Clusters from Cloud Shell

Lab: Upgrading Google Kubernetes Engine Clusters

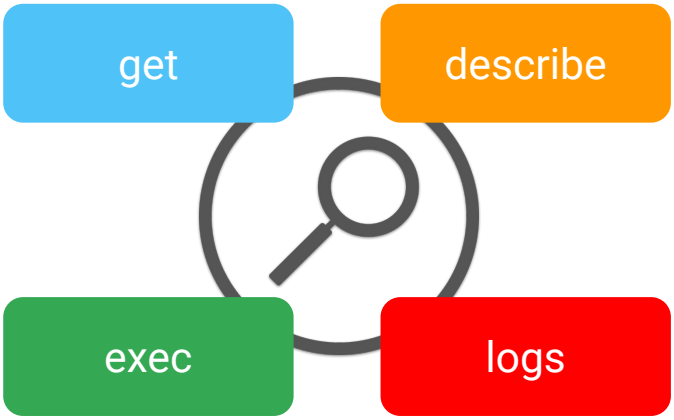
Summary



Google Cloud

In the previous lesson, we discussed the Command part of kubectl's syntax. It's the way to specify the action you want to perform. When your application is running, you'll inevitably need to debug problems with it. Introspection is the act of gathering information about the containers, Pods, Services, and other engines running within the cluster. In this lesson, we'll revisit kubectl's syntax and discuss its get, describe, exec, and logs commands in more detail.

Use kubectl to gather info about your app



Google Cloud

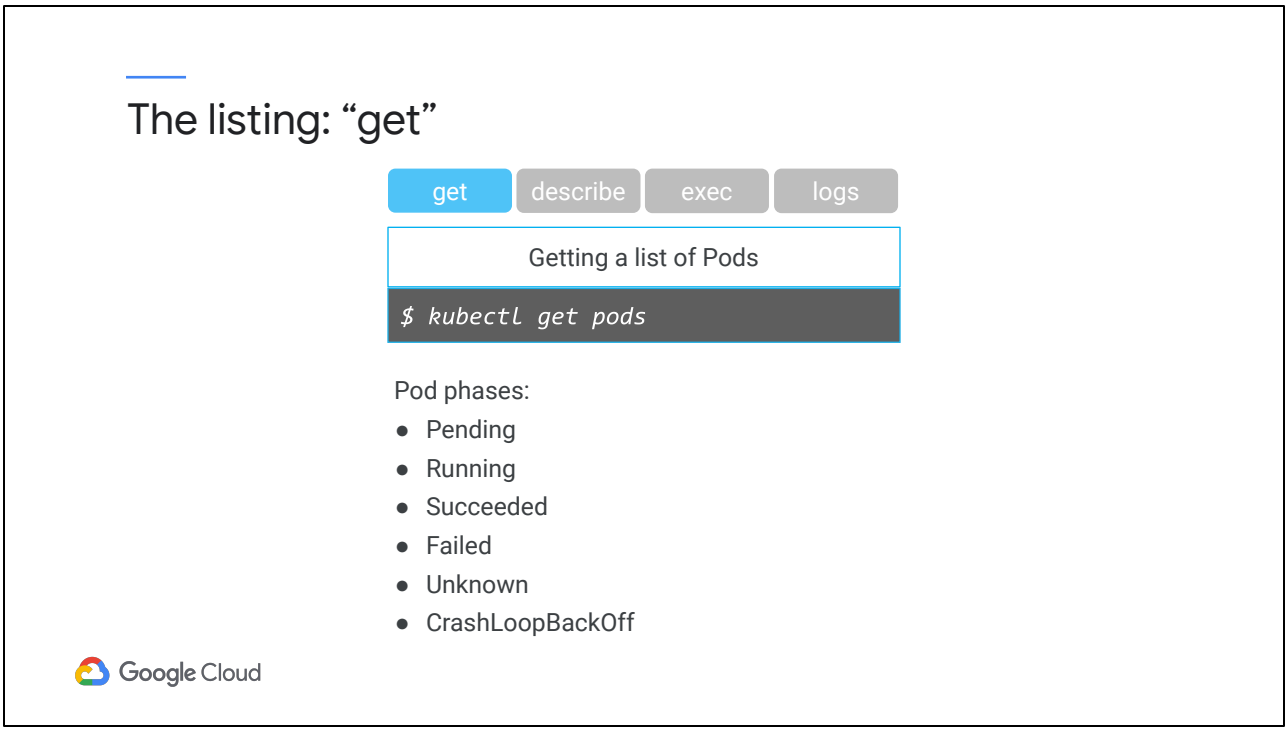
You can retrieve a list of objects using a command like ‘kubectl **get** pods’, which returns a list of all the Pods in the cluster and tells you their status.

For more information on a specific Pod, run the command ‘kubectl **describe** my-pod-name’, which gives you detailed information about the Pod named “my-pod-name”.

You can even test and debug within your Pod using the “kubectl **exec** my-pod-name” command to execute commands and applications.

And the **logs** command provides a powerful tool to see what is happening inside a Pod. Logs are always useful in troubleshooting; the logs command allows you to quickly and easily view errors or debugging messages written out by the applications running inside Pods. The logs command is particularly useful when you need to find out more information about containers that are failing to run successfully.

Now, let’s focus on troubleshooting Kubernetes and GKE. A good place to start is with Pods, the basic units of Kubernetes.



A simple kubectl ‘get pods’ command tells you whether your Pod is running. It shows the Pod’s phase status as *Pending, Running, Succeeded, Failed* or *Unknown*. A Pod’s phase provides a high-level summary of its lifecycle, not the comprehensive details about a Pod or its containers.

Pending means a Pod has been accepted by Kubernetes, but it’s still being scheduled. This means that the container images that are defined for the Pod have not yet been created by the container runtime. For example, when images are being pulled from the repository, the Pod will be in the *Pending* phase.

A Pod is *Running* when it has successfully attached to a node and all of its containers have been created. Containers inside a Pod can be starting, restarting, or running continuously.

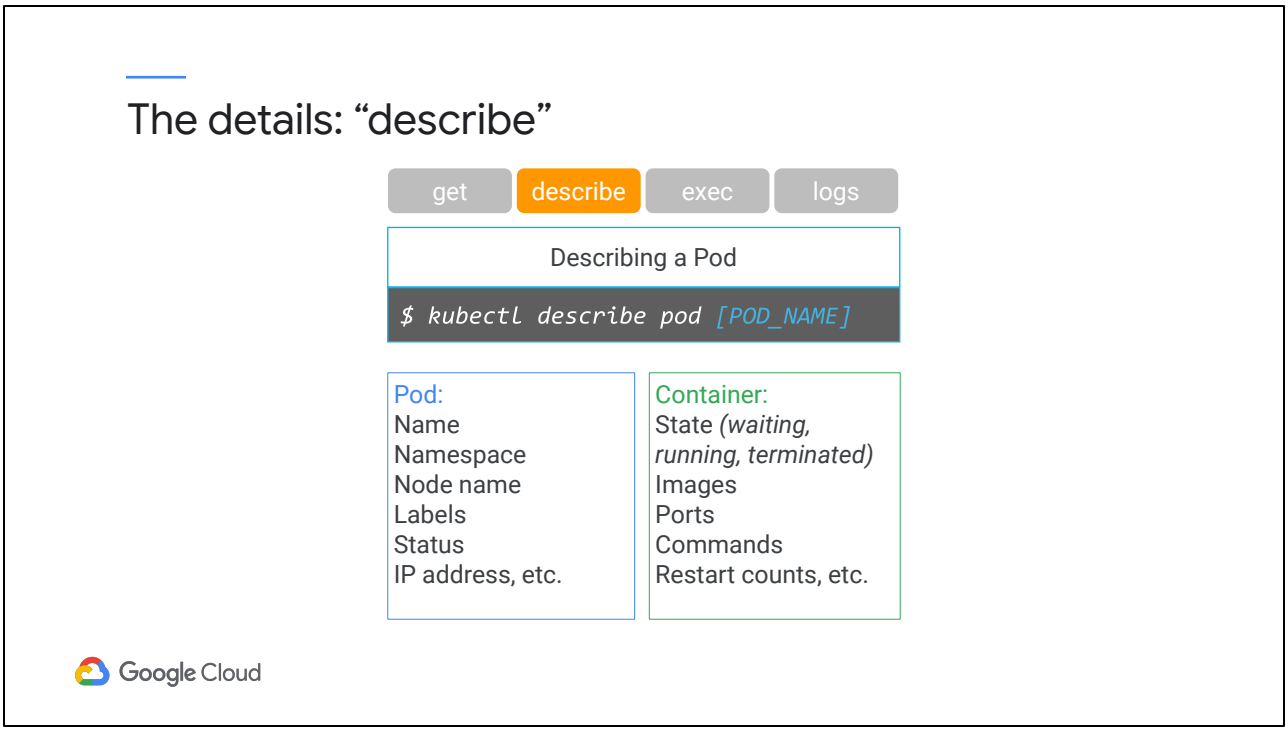
Succeeded means that all containers have finished running successfully. In other words, they’ve terminated successfully and they won’t be restarting.

Failed means a container has terminated with a failure, and it won’t be restarting.

Unknown is where the state of the Pod simply cannot be retrieved, probably because of a communication error between the control plane and a kubelet. It’s not a

commonly seen state.

CrashLoopBackOff means that one of the containers in the Pod exited unexpectedly even after it was restarted at least once. This is a common error. Usually, CrashLoopBackOff means that the Pod isn’t configured correctly.



To investigate a Pod in more detail, use the kubectl ‘describe pod’ command to fetch details.

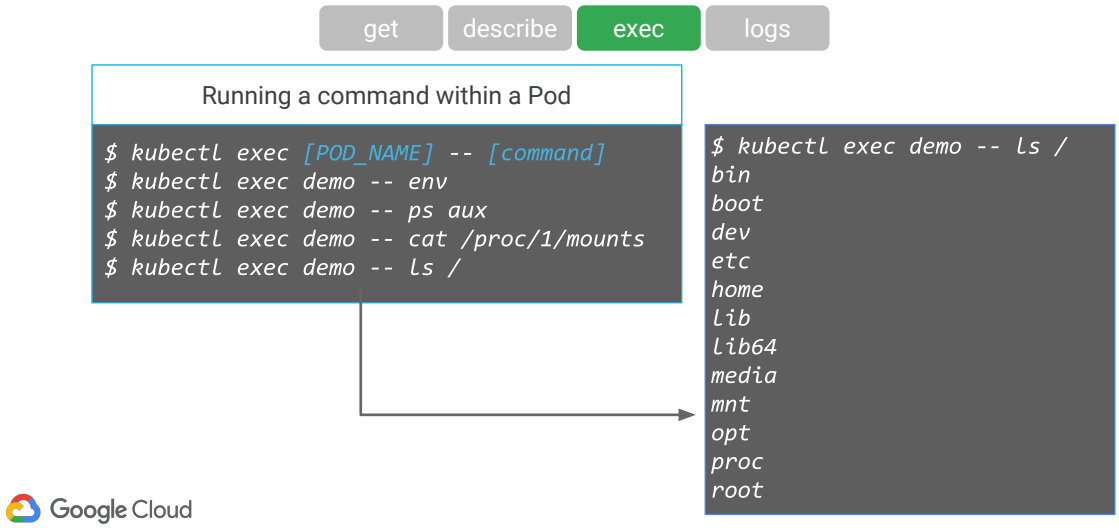
This gives you information about a Pod and its containers such as labels, resource requirements, and volumes. It also details the status information about the container and Pod, such as state, readiness, restart count, and events.

A container state can be either *Waiting*, *Running*, or *Terminated*.

You can view restart counts and configuration options like the restartPolicy. For example, with restartPolicy set to Always, the container will restart continuously.

At the end of the output, you can review events. Imagine a Pod and its container running successfully. You then attempt to update the container image to a newer version, but the new container image fails to download. In this case, the Pod will change to *Pending* phase, and the container state will become a *Waiting* state, with the reason being “ImagePullBackOff”.

Interaction: “exec”



If you need to troubleshoot an application, you can run a single command in the container, or if you need to do more than that you can get a shell in the container. You might need to troubleshoot connectivity or another application for example.

Single command execution using the exec command as shown here allows you to run a single command inside a container and view the results in your own command shell. This is useful when a single command, such as ping, will do.

Working inside a Pod

Running a command within a Pod

```
$ kubectl exec -it [POD_NAME] -- [command]
```

```
$ kubectl exec -it demo -- /bin/bash
root@demo:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt
proc  root  run  sbin  srv  sys  tmp  usr  var
root@demo:/#
```

Pass terminal's standard input (stdin) to the container.

Display the container's standard output (stdout) in your terminal window.

Uses the flags -i and -t (or -it).

Google Cloud

If you need to install a package, like a network monitoring tool or a text editor, before you can begin troubleshooting, you can launch an interactive shell using the -it switch which connects your shell to the container so that you can work inside the container.

Note that it isn't a best practice to install software directly into a container. Changes made by containers to their file systems are usually ephemeral. You should build container images that have exactly the software you need, instead of temporarily repairing them at run time. The interactive shell will allow you to figure out what needs to be changed to solve a problem, but you should then integrate those changes into your container images and redeploy them.

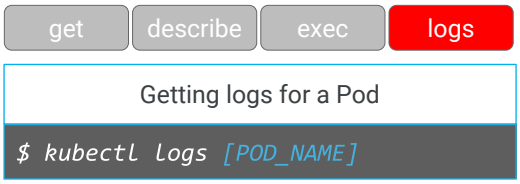
This syntax attaches the standard input and standard output of the container to your terminal window or command shell.

The -i argument tells kubectl to pass the terminal's standard input to the container, and the -t argument tells kubectl that the input is a TTY.

If you don't use these arguments then the exec command will be executed in the remote container and return immediately to your local shell.

If a Pod has more than one container use the -c argument to specify the specific container to attach to in the Pod.

The history: “logs”



The Pod logs include:

- `stdout`: Standard output on the console
- `stderr`: Error messages

Google Cloud

You can also use the kubectl command to view the logs of a Pod. If the Pod has multiple containers, you can use the -c argument to show the logs for a specific container inside the Pod.

The logs contain both the standard output and standard error messages that the applications within the container have generated.

Agenda

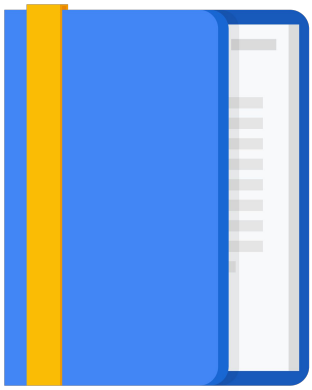
The kubectl command

Introspection

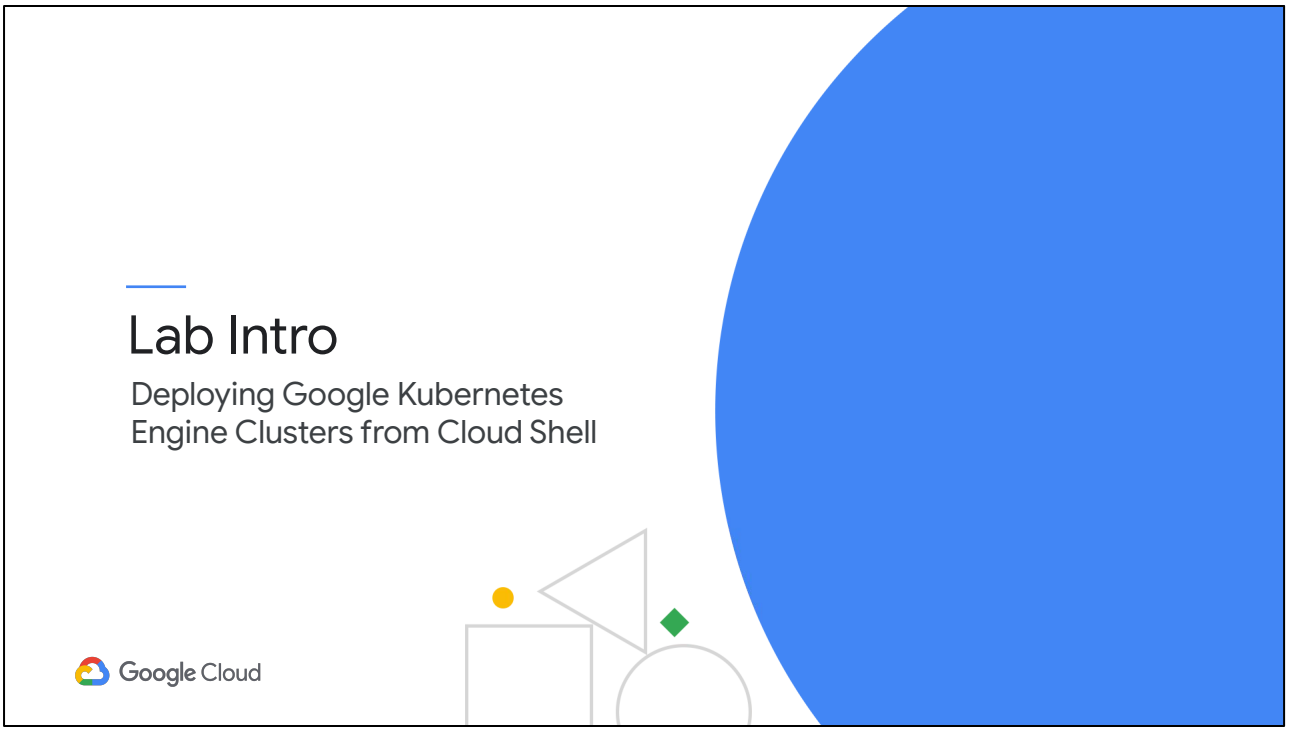
Lab: Deploying Google Kubernetes Engine Clusters from Cloud Shell

Lab: Upgrading Google Kubernetes Engine Clusters

Summary



Google Cloud



In this lab, you'll use the command line to build GKE clusters. You'll inspect the kubeconfig file and use kubectl to manipulate the cluster.

The tasks that you'll learn to perform include using Cloud Shell to deploy GKE clusters, modifying the number of nodes in a GKE cluster, and authenticating to a GKE cluster. You'll then use the Cloud Shell code editor to inspect the kubectl configuration files. You'll also use Cloud Shell and kubectl to inspect a GKE cluster and use Cloud Shell to deploy Pods to GKE clusters. Lastly, you'll introspect GKE Pods by connecting to a Pod to adjust settings, edit files, and make other live changes to the Pod.

Agenda

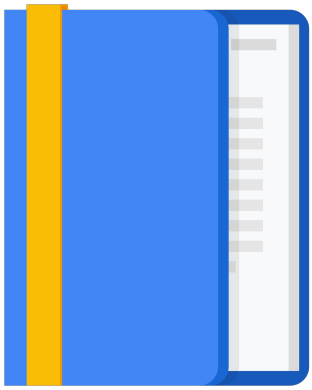
The kubectl command

Introspection

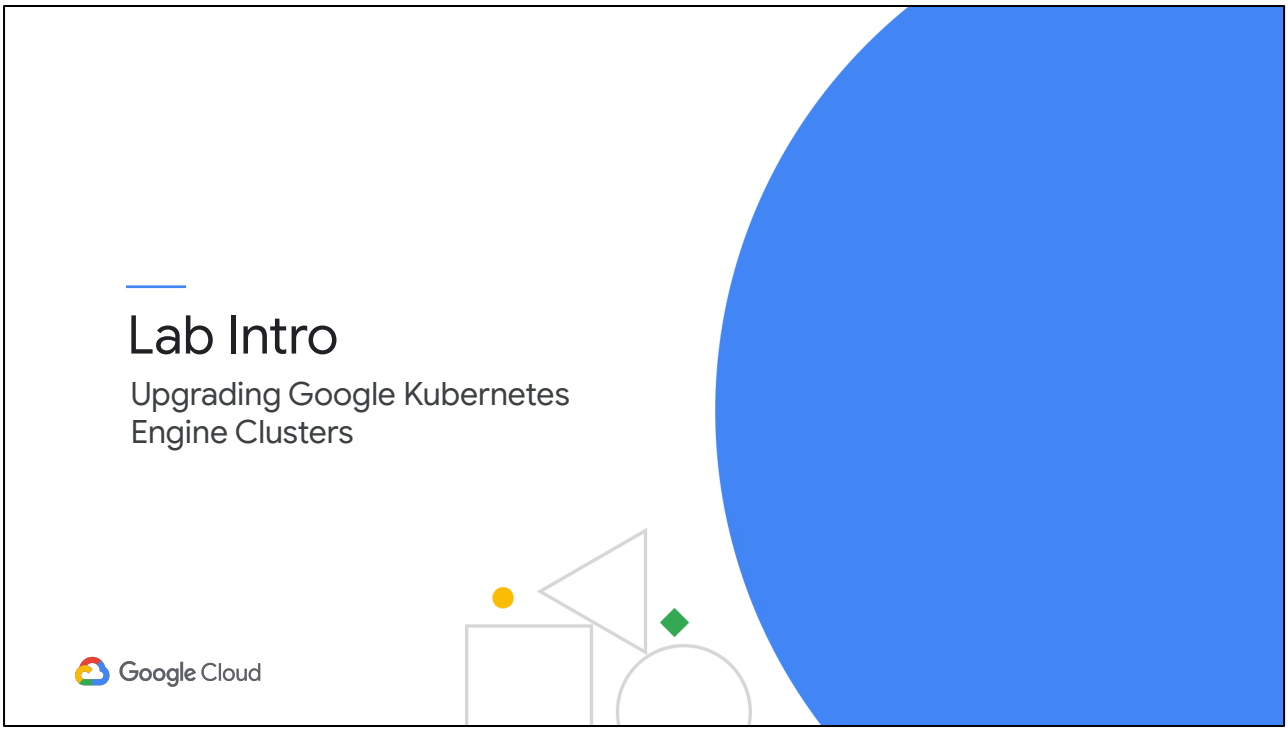
Lab: Deploying Google Kubernetes Engine Clusters from Cloud Shell

Lab: Upgrading Google Kubernetes Engine Clusters

Summary



Google Cloud



In this lab, you'll learn how to use the Cloud Console to upgrade your GKE cluster. Your first task will be to use Cloud Shell to deploy a GKE cluster. You'll then upgrade your GKE cluster, which will include upgrading the nodes of your cluster to the same version as the control plane.

Agenda

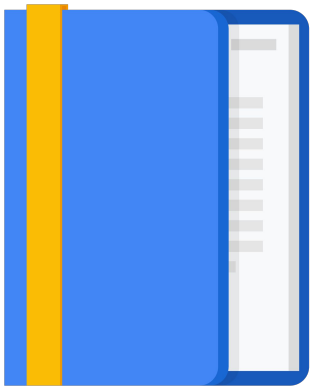
The kubectl command

Introspection

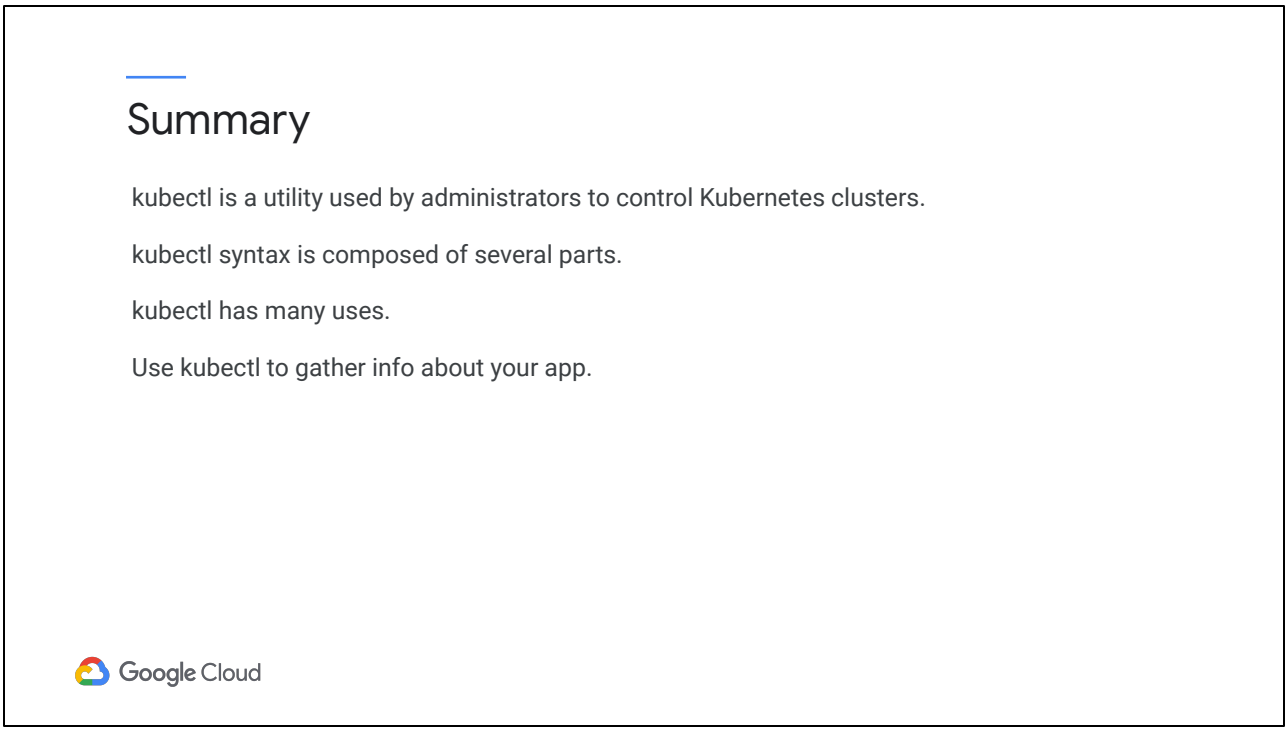
Lab: Deploying Google Kubernetes Engine Clusters from Cloud Shell

Lab: Upgrading Google Kubernetes Engine Clusters

Summary



Google Cloud



That concludes the Kubernetes Operations module. Let me recap some of the key learning points.

Kubectl transforms your command-line entries into API calls that it sends to the Kube API server within your selected Kubernetes cluster.
Before it can do any work for you, kubectl must be configured with the location and credentials of a Kubernetes cluster.

Kubectl syntax is composed of several parts: the command, the type, the name, and optional flags. the ‘COMMAND’ specifies the action that you want to perform, such as get, describe, logs, or exec. ‘TYPE’ defines the Kubernetes object that the ‘command’ acts on, and ‘NAME’ specifies the object defined in ‘TYPE.’ Some commands support additional optional flags that you can include at the end of the command.

You can use the kubectl command to create, view, interact with and delete Kubernetes objects.

The ‘kubectl get pods’ command returns a list of all the Pods in the cluster and tells you their status.
‘kubectl describe my-pod-name’ gives you detailed information about the named Pod.
‘kubectl exec my-pod-name’ allows you to test and debug within your named Pod.
The ‘kubectl logs’ command allows you to view the logs of a Pod.


Google Cloud