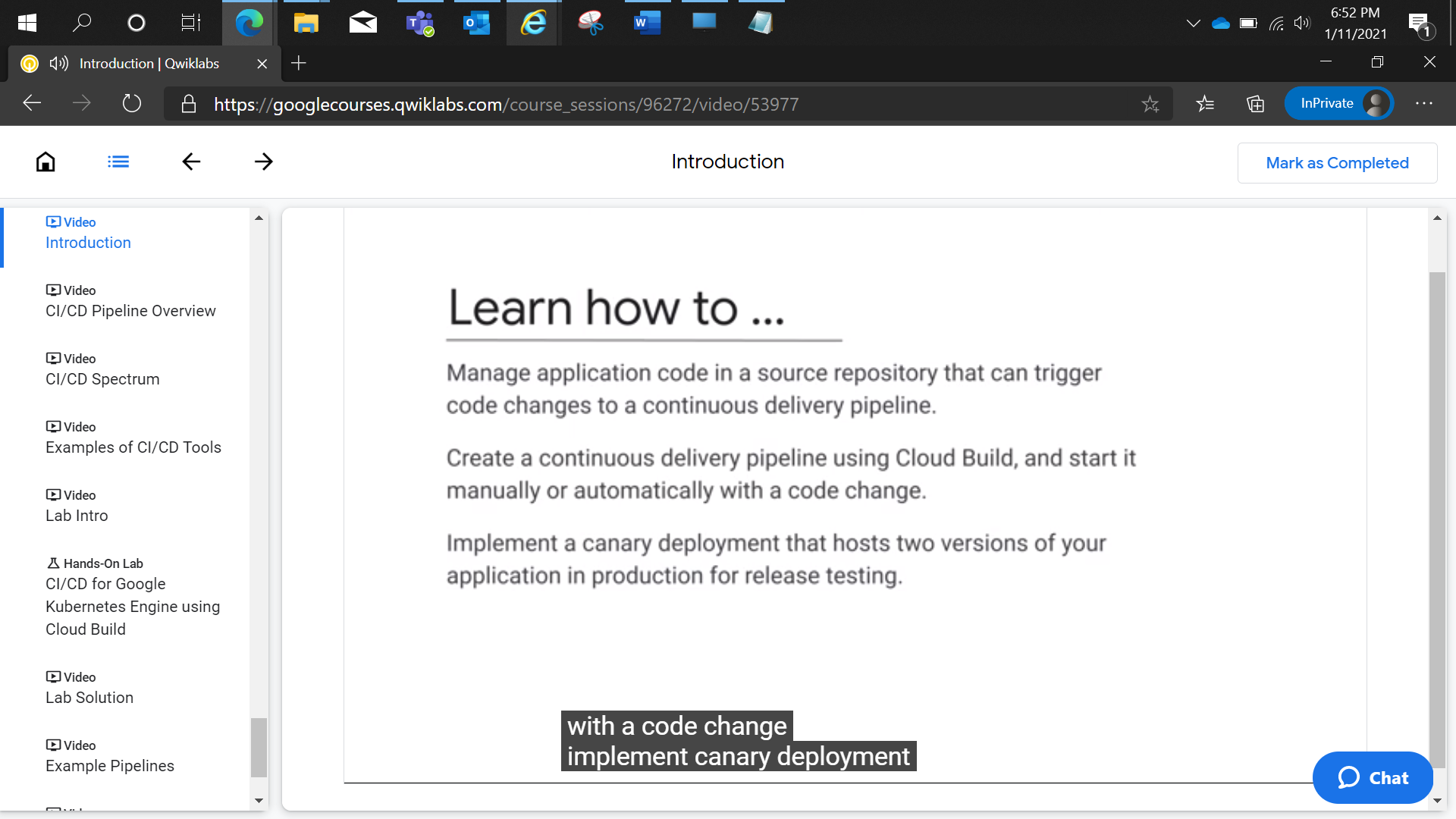
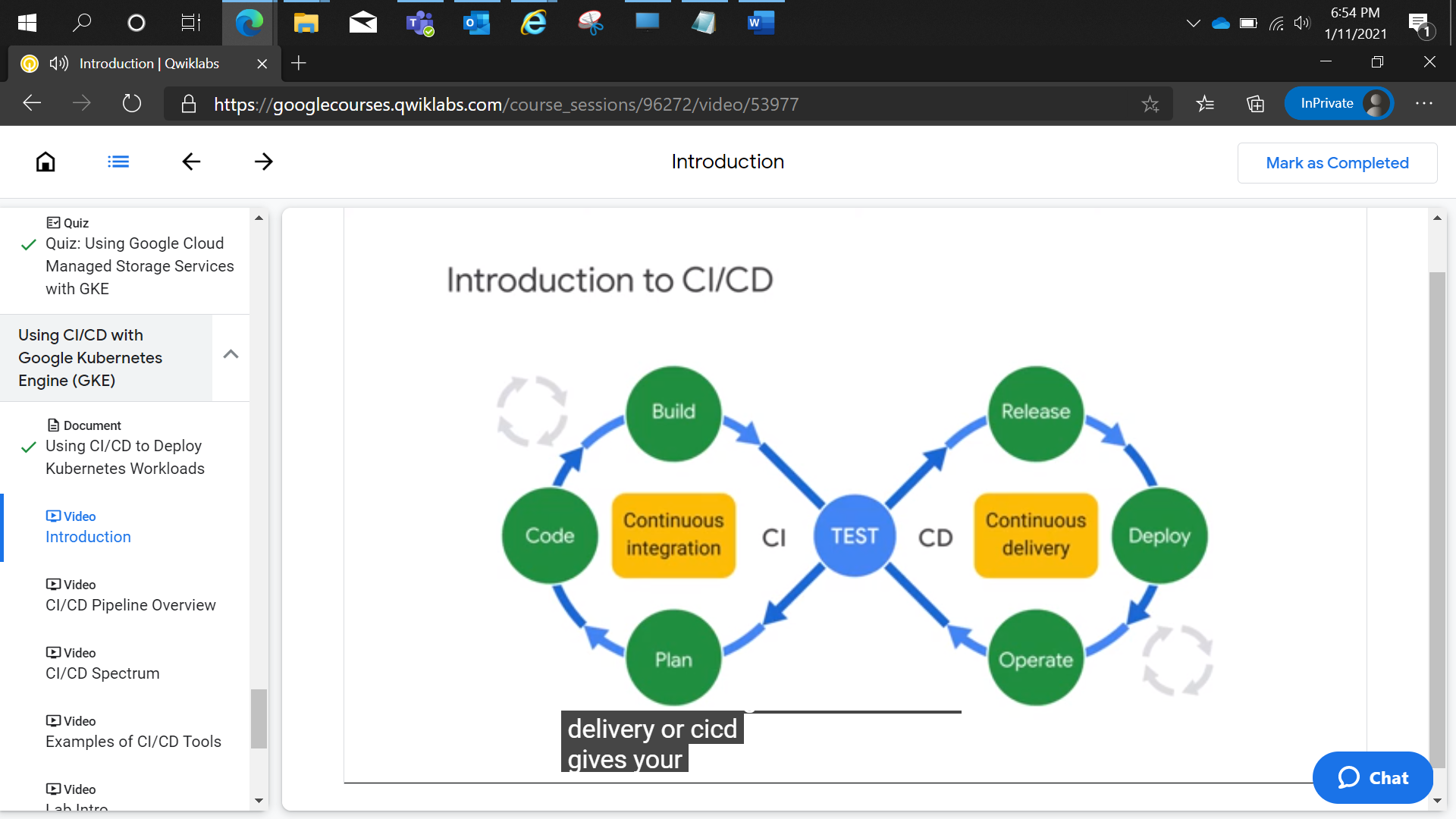
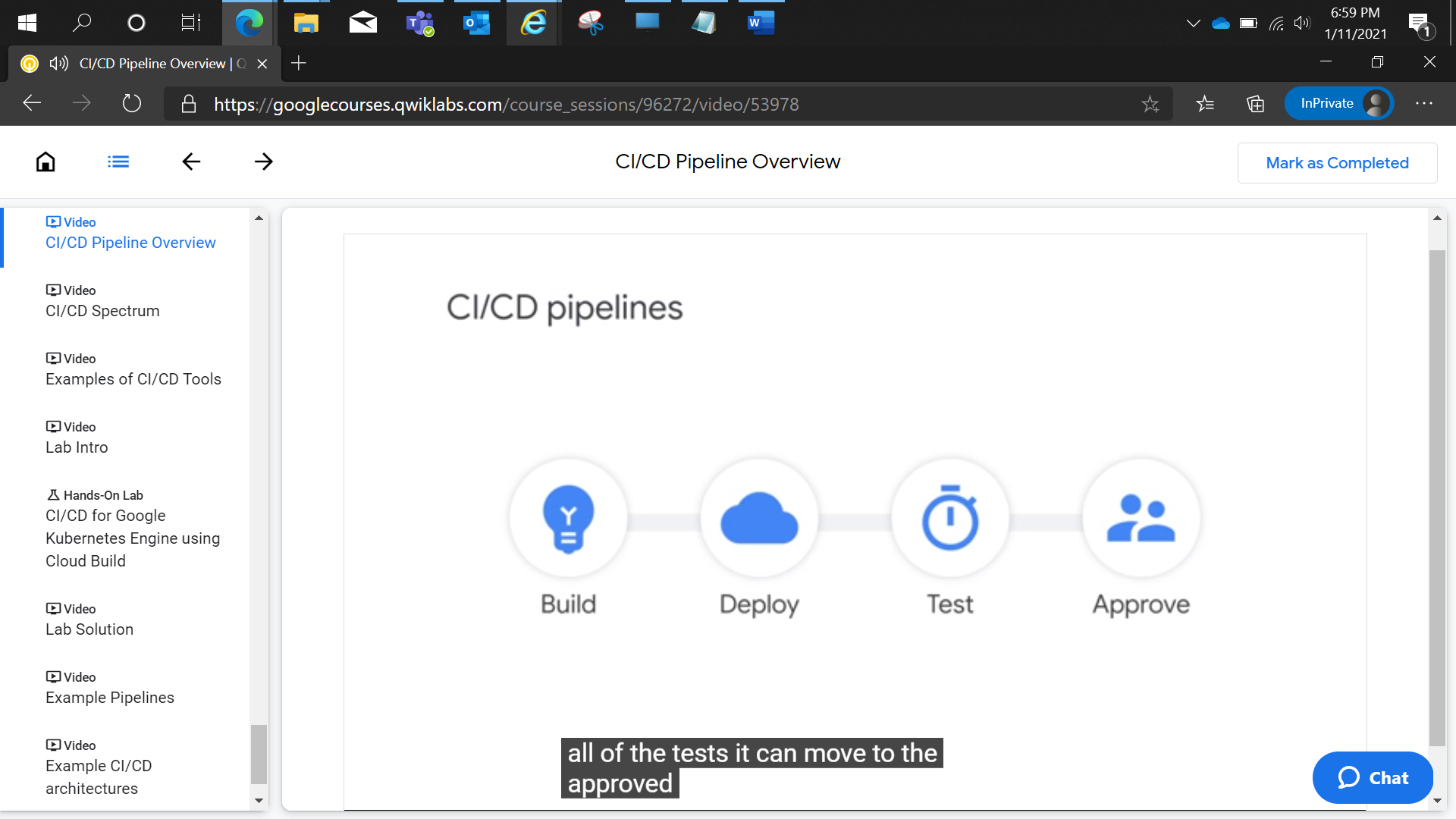
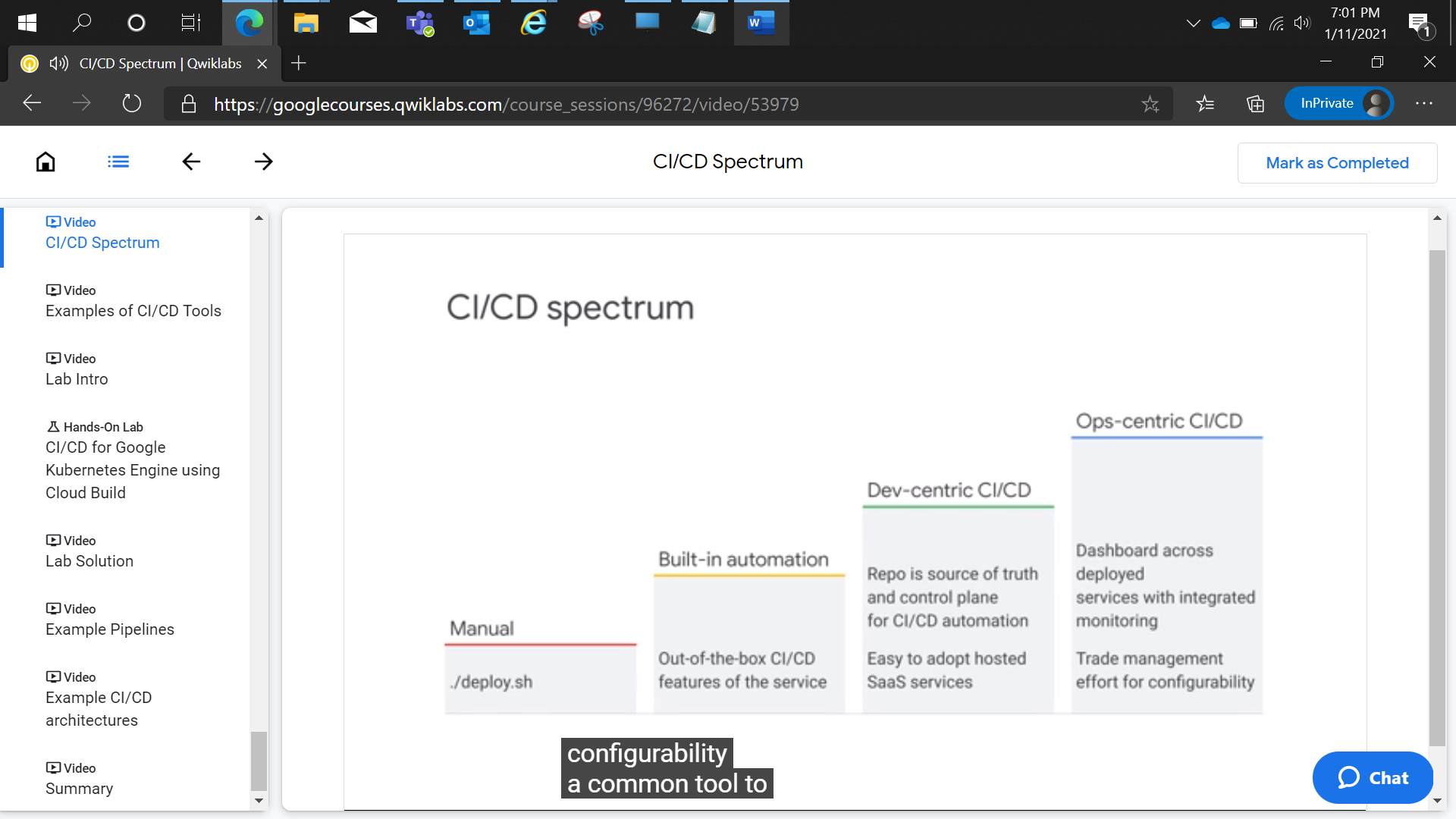
Chapter 2.3.4 Using CI/CD with GKE

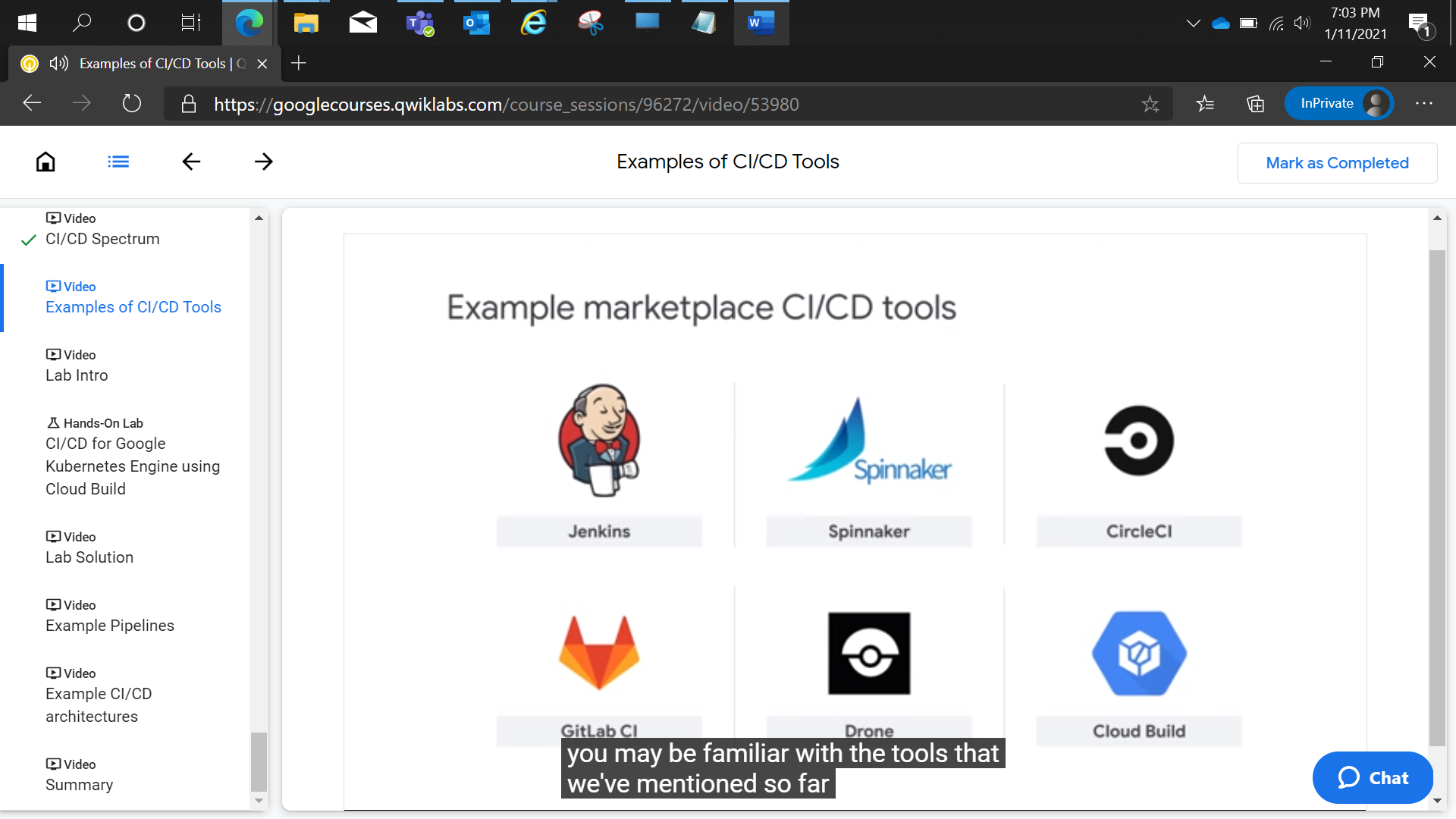


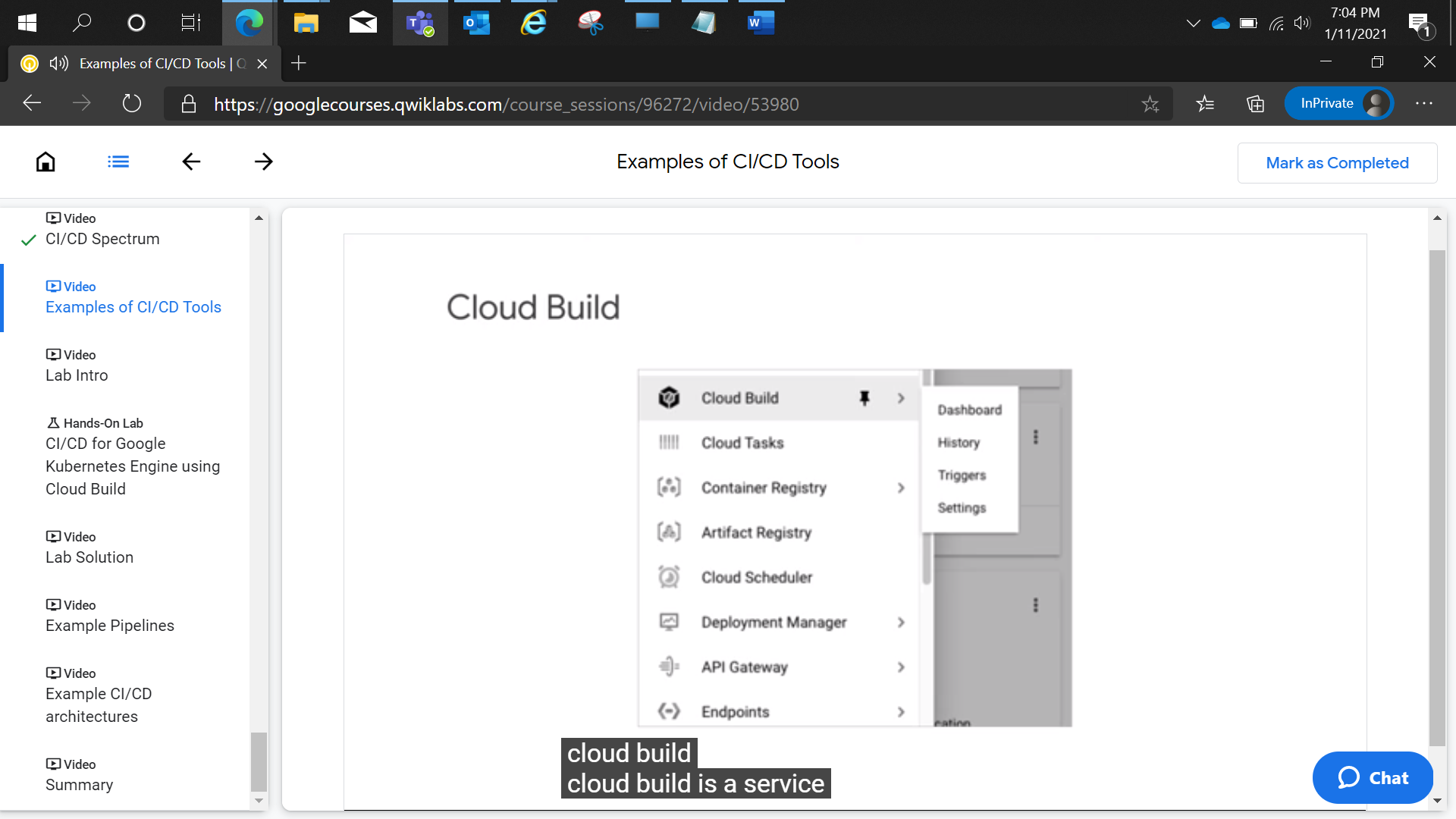


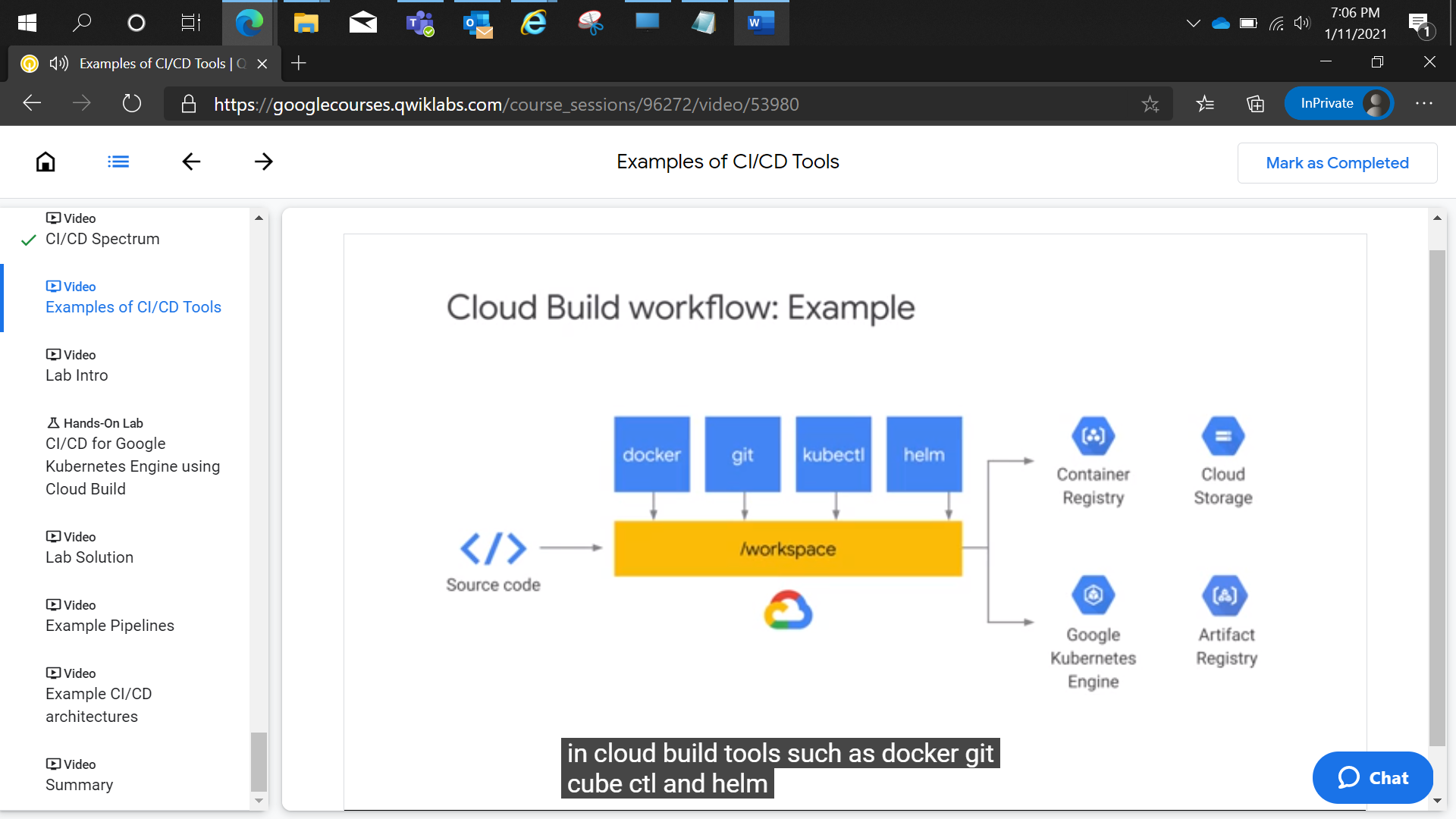












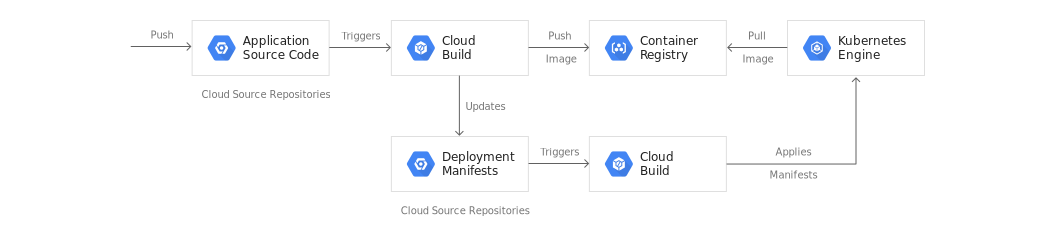
CI/CD for Google Kubernetes Engine using Cloud Build

1 hour 30 minutesFree

Rate Lab

**Overview**

In this lab, you create a CI/CD pipeline that automatically builds a container image from committed code, stores the image in Container Registry, updates a Kubernetes manifest in a Git repository, and deploys the application to Google Kubernetes Engine using that manifest.



For this lab you will create 2 Git repositories:

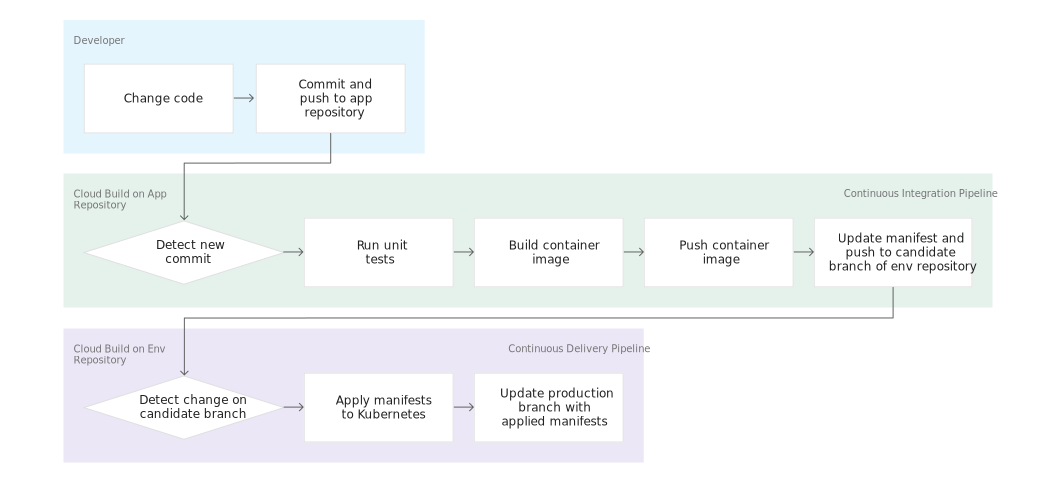
* app repository: contains the source code of the application itself
* env repository: contains the manifests for the Kubernetes Deployment

When you push a change to the **app repository**, the Cloud Build pipeline runs tests, builds a container image, and pushes it to Container Registry. After pushing the image, Cloud Build updates the Deployment manifest and pushes it to the **env repository**. This triggers another Cloud Build pipeline that applies the manifest to the GKE cluster and, if successful, stores the manifest in another branch of the **env repository**.

We keep the app and env repositories separate because they have different lifecycles and uses. The main users of the **app repository** are actual humans and this repository is dedicated to a specific application. The main users of the **env repository** are automated systems (such as Cloud Build), and this repository might be shared by several applications. The **env repository** can have several branches that each map to a specific environment (you only use production in this lab) and reference a specific container image, whereas the **app repository** does not.

When you finish this lab, you have a system where you can easily:

* Distinguish between failed and successful deployments by looking at the Cloud Build history,
* Access the manifest currently used by looking at the production branch of the **env repository**,
* Rollback to any previous version by re-executing the corresponding Cloud Build build.



**Note:**

This lab uses Cloud Build to execute the pipelines, but there are other popular build automation tools that could serve as alternatives, such as Spinnaker and Jenkins. At this time, some of these other tools have superior support for blue/green deployments, canary analysis, and similar features that may be required in more advanced CI/CD implementations.

**Objectives**

In this lab, you learn how to perform the following tasks:

* Create Kubernetes Engine clusters
* Create Cloud Source Repositories
* Trigger Cloud Build from Cloud Source Repositories
* Automate tests and publish a deployable container image via Cloud Build
* Manage resources deployed in a Kubernetes Engine cluster via Cloud Build

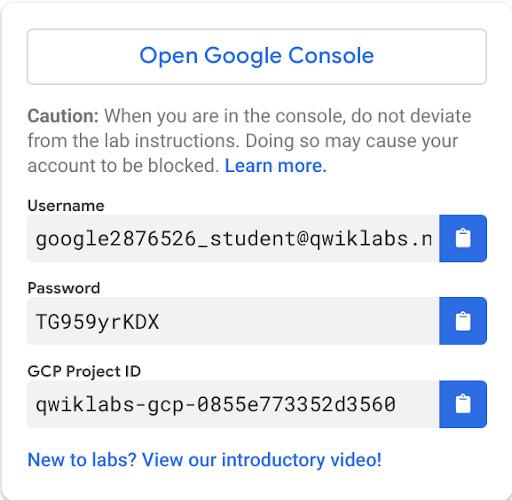
**Task 0. Lab Setup**

**Access Qwiklabs**

For each lab, you get a new GCP project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example,  and make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

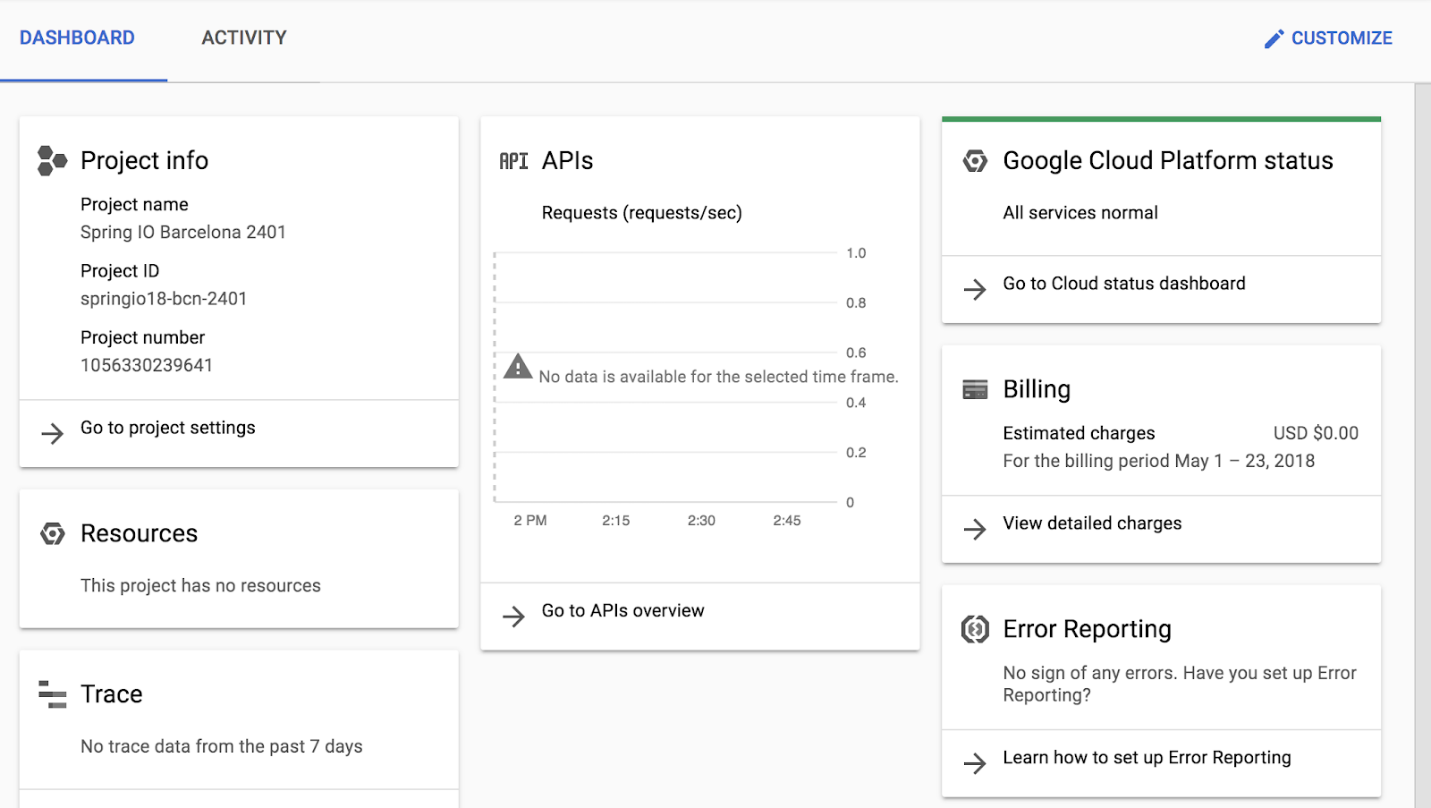
1. When ready, click .
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

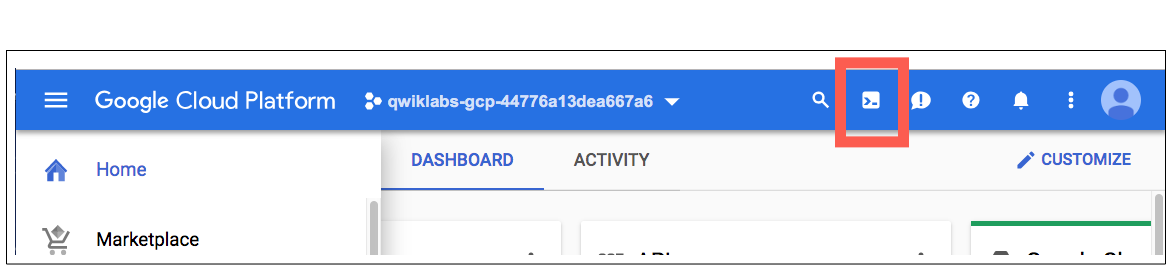
After you complete the initial sign-in steps, the project dashboard appears.

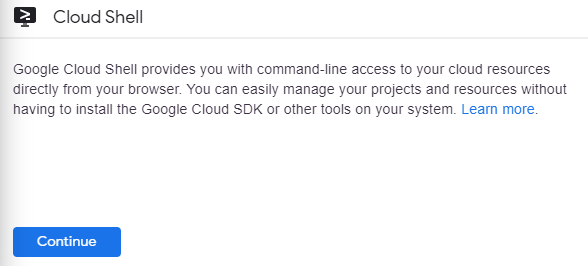


Activate Google Cloud Shell

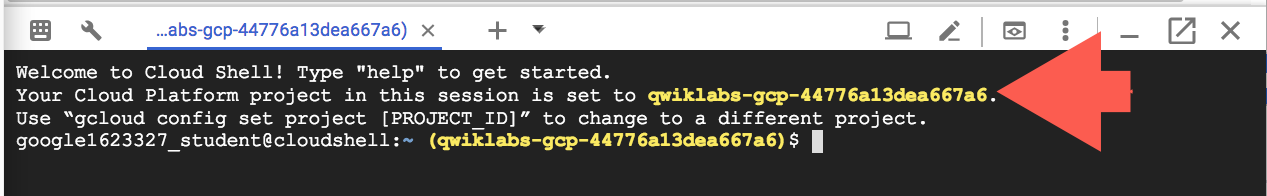
Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**. 

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



**gcloud** is the command-line tool for Google Cloud Platform. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

Example output:

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

Output:

[core]

project = <project\_ID>content\_copy

Example output:

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

Full documentation of **gcloud** is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud).

**Task 1. Initialize Your Lab**

In this task you will prepare your Google Cloud Project for use by enabling the required APIs, initializing the git configuration in Cloud Shell, and downloading the sample code used later in the lab.

1. In Cloud Shell, run the following command to enable the APIs for GKE, Cloud Build, Cloud Source Repositories and Container Analysis:

gcloud services enable container.googleapis.com \

cloudbuild.googleapis.com \

sourcerepo.googleapis.com \

containeranalysis.googleapis.com

content\_copy

1. In Cloud Shell, create a GKE cluster that you will use to deploy the sample application of this lab.

gcloud container clusters create hello-cloudbuild \

--num-nodes 1 --zone us-central1-b

content\_copy

1. If you have never used Git in Cloud Shell, configure it with your name and email address. Git will use those to identify you as the author of the commits you will create in Cloud Shell.

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

content\_copy

Click *Check my progress* to verify the objective.

Initialize Your Lab

Check my progress

**Task 2. Create the Git repositories in Cloud Source Repositories**

In this task, you create the two Git repositories (**hello-cloudbuild-app** and **hello-cloudbuild-env**) used in this lab, and initialize **hello-cloudbuild-app** with some sample code.

1. In Cloud Shell, create the two Git repositories.

gcloud source repos create hello-cloudbuild-app

gcloud source repos create hello-cloudbuild-env

content\_copy

1. Clone the sample code from GitHub.

cd ~

git clone https://github.com/GoogleCloudPlatform/gke-gitops-tutorial-cloudbuild \

hello-cloudbuild-app

content\_copy

1. Configure Cloud Source Repositories as a remote.

cd ~/hello-cloudbuild-app

PROJECT\_ID=$(gcloud config get-value project)

git remote add google \

"https://source.developers.google.com/p/${PROJECT\_ID}/r/hello-cloudbuild-app"

content\_copy

The code you just cloned contains a simple "Hello World" application.

from flask import Flask

app = Flask('hello-cloudbuild')

@app.route('/')

def hello():

return "Hello World!\n"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host = '0.0.0.0', port = 8080)content\_copy

Click *Check my progress* to verify the objective.

Create the Git repositories in Cloud Source Repositories

Check my progress

**Task 3. Create a container image with Cloud Build**

The code you cloned already contains the following Dockerfile.

FROM python:3.7-slim

RUN pip install flask

WORKDIR /app

COPY app.py /app/app.py

ENTRYPOINT ["python"]

CMD ["/app/app.py"]content\_copy

With this Dockerfile, you can create a container image with Cloud Build and store it in Container Registry.

1. In Cloud Shell, create a Cloud Build build based on the latest commit with the following command.

cd ~/hello-cloudbuild-app

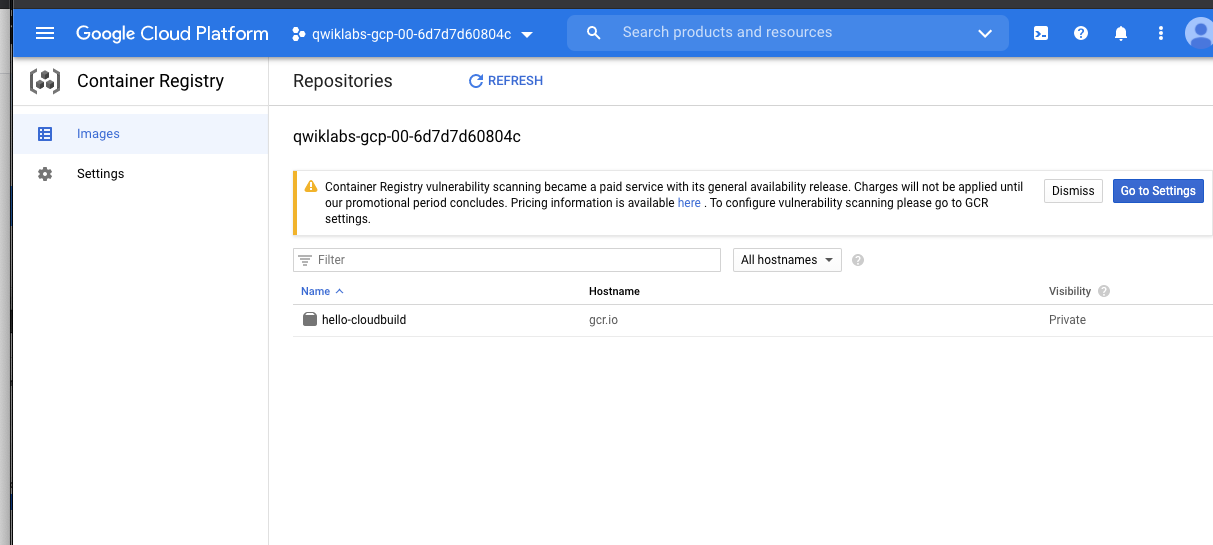
COMMIT\_ID="$(git rev-parse --short=7 HEAD)"

gcloud builds submit --tag="gcr.io/${PROJECT\_ID}/hello-cloudbuild:${COMMIT\_ID}" .

content\_copy

Cloud Build streams the logs generated by the creation of the container image to your terminal when you execute this command.

1. After the build finishes, in the Google Cloud Console **Container Registry > Images** to verify that your new container image is indeed available by in Container Registry.



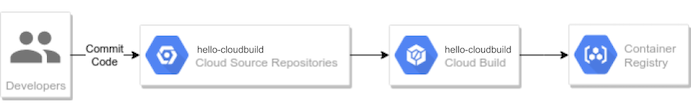
Click *Check my progress* to verify the objective.

Create a container image with Cloud Build

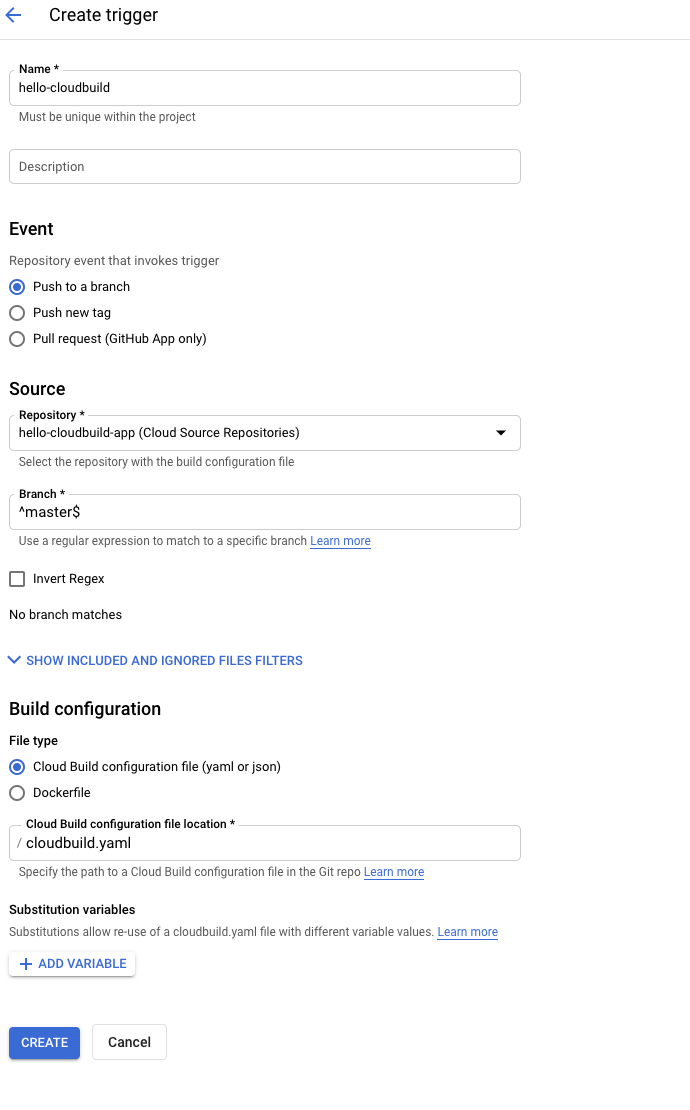
Check my progress

**Task 4. Create the Continuous Integration (CI) Pipeline**

In this task, you will configure Cloud Build to automatically run a small unit test, build the container image, and then push it to Container Registry. Pushing a new commit to Cloud Source Repositories triggers automatically this pipeline. The **cloudbuild.yaml** file already included in the code is the pipeline's configuration.



1. In the Google Cloud Console, **Cloud Build > Triggers**.
2. Click **Create Trigger**
3. In the Name field, type **hello-cloudbuild**.
4. Under **Event**, select **Push to a branch**.
5. Under **Source**, select **hello-cloudbuild-app** as your **Repository** and **^master$** as your **Branch**.
6. Under **Build configuration**, select **Cloud Build configuration file**.
7. In the **Cloud Build configuration file location** field, type **cloudbuild.yaml** after the /.
8. Click **Create**



When the trigger is created, return to the Cloud Shell. You now need to push the application code to Cloud Source Repositories to trigger the CI pipeline in Cloud Build. To start this trigger, run the following command

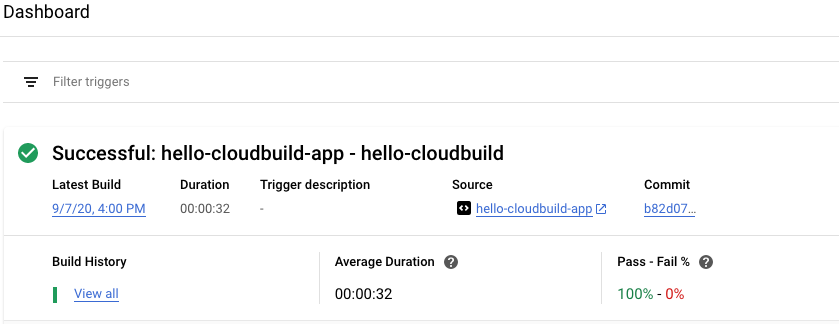
cd ~/hello-cloudbuild-app

git push google master

content\_copy

In the Google Cloud Console, **Cloud Build > Dashboard**.

You should see a build running or having recently finished. You can click on the build to follow its execution and examine its logs.



Click *Check my progress* to verify the objective.

Create the Continuous Integration (CI) Pipeline

Check my progress

**Task 5. Create the Test Environment and CD Pipeline**

Cloud Build is also used for the continuous delivery pipeline. The pipeline runs each time a commit is pushed to the candidate branch of the \_\_hello-cloudbuild-env\_\_repository. The pipeline applies the new version of the manifest to the Kubernetes cluster and, if successful, copies the manifest over to the production branch. This process has the following properties:

* The candidate branch is a history of the deployment attempts.
* The production branch is a history of the successful deployments.
* You have a view of successful and failed deployments in Cloud Build.
* You can rollback to any previous deployment by re-executing the corresponding build in Cloud Build. A rollback also updates the production branch to truthfully reflect the history of deployments.

You will modify the continuous integration pipeline to update the candidate branch of the **hello-cloudbuild-env** repository, triggering the continuous delivery pipeline.

**Grant Cloud Build access to GKE**

To deploy the application in your Kubernetes cluster, Cloud Build needs the Kubernetes Engine Developer Identity and Access Management Role.

1. In Cloud Shell execute the following command:

PROJECT\_NUMBER="$(gcloud projects describe ${PROJECT\_ID} --format='get(projectNumber)')"

gcloud projects add-iam-policy-binding ${PROJECT\_NUMBER} \

--member=serviceAccount:${PROJECT\_NUMBER}@cloudbuild.gserviceaccount.com \

--role=roles/container.developer

content\_copy

**Initialize the hello-cloudbuild-env repository**

You need to initialize the **hello-cloudbuild-env** repository with two branches (production and candidate) and a Cloud Build configuration file describing the deployment process.

The first step is to clone the **hello-cloudbuild-env** repository and create the production branch. It is still empty.

1. In Cloud Shell execute the following command:

cd ~

gcloud source repos clone hello-cloudbuild-env

cd ~/hello-cloudbuild-env

git checkout -b production

content\_copy

1. Next you need to copy the **cloudbuild-delivery.yaml** file available in the *hello-cloudbuild-app* repository and commit the change.

cd ~/hello-cloudbuild-env

cp ~/hello-cloudbuild-app/cloudbuild-delivery.yaml ~/hello-cloudbuild-env/cloudbuild.yaml

git add .

git commit -m "Create cloudbuild.yaml for deployment"

content\_copy

**Note:**

The cloudbuild-delivery.yaml file describes the deployment process to be run in Cloud Build. It has two steps:

* Cloud Build applies the manifest on the GKE cluster.
* If successful, Cloud Build copies the manifest on the production branch.

1. Create a candidate branch and push both branches for them to be available in Cloud Source Repositories.

git checkout -b candidate

git push origin production

git push origin candidate

content\_copy

1. Grant the Source Repository Writer IAM role to the Cloud Build service account for the **hello-cloudbuild-env** repository.

PROJECT\_NUMBER="$(gcloud projects describe ${PROJECT\_ID} \

--format='get(projectNumber)')"

cat >/tmp/hello-cloudbuild-env-policy.yaml <<EOF

bindings:

- members:

- serviceAccount:${PROJECT\_NUMBER}@cloudbuild.gserviceaccount.com

role: roles/source.writer

EOF

gcloud source repos set-iam-policy \

hello-cloudbuild-env /tmp/hello-cloudbuild-env-policy.yaml

content\_copy

Click *Check my progress* to verify the objective.

Grant Kubernetes Engine Developer and Source Repository Writer IAM role to cloud build

Check my progress

**Create the trigger for the continuous delivery pipeline**

1. In the Google Cloud Console, **Cloud Build > Triggers**.
2. Click **Create Trigger**
3. In the Name field, type **hello-cloudbuild-deploy**.
4. Under **Event**, select **Push to a branch**.
5. Under **Source**, select **hello-cloudbuild-env** as your **Repository** and **^candidate$** as your **Branch**.
6. Under **Build configuration**, select **Cloud Build configuration file**.
7. In the **Cloud Build configuration file location** field, type **cloudbuild.yaml** after the /.
8. Click **Create**



Modify the continuous integration pipeline to trigger the continuous delivery pipeline

In this section, you add some steps to the continuous integration pipeline that will generate a new version of the Kubernetes manifest and push it to the \_\_hello-cloudbuild-env\_\_repository to trigger the continuous delivery pipeline.

1. Copy the extended version of the **cloudbuild.yaml** file for the **app repository**.

cd ~/hello-cloudbuild-app

cp cloudbuild-trigger-cd.yaml cloudbuild.yaml

content\_copy

The **cloudbuild-trigger-cd.yaml** is an extended version of the **cloudbuild.yaml** file. It adds the steps below: they generate the new Kubernetes manifest and trigger the continuous delivery pipeline.

**Note:**

This pipeline uses a simple sed to render the manifest template. In reality, you will benefit from using a dedicated tool such as kustomize or skaffold. They allow for more control over the rendering of the manifest templates.

1. Commit the modifications and push them to Cloud Source Repositories.

cd ~/hello-cloudbuild-app

git add cloudbuild.yaml

git commit -m "Trigger CD pipeline"

git push google master

content\_copy

This triggers the continuous integration pipeline in Cloud Build.

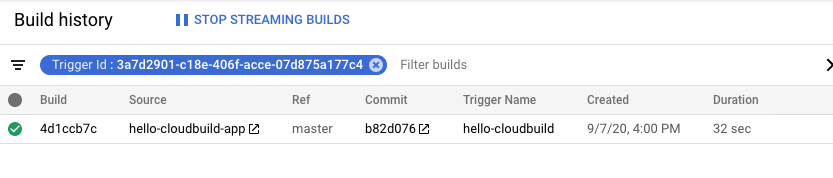
Click *Check my progress* to verify the objective.

Create the trigger for the continuous delivery pipeline

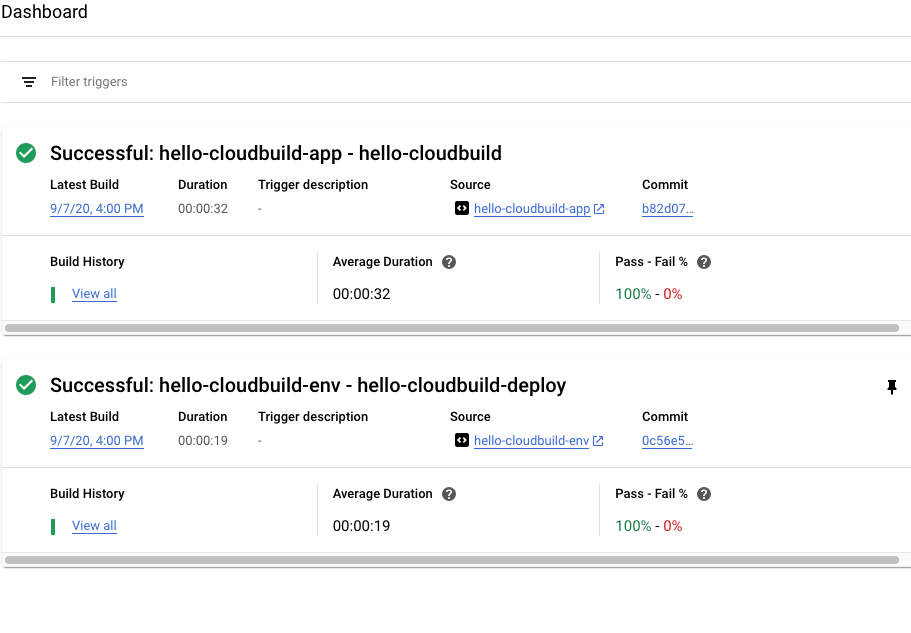
Check my progress

**Task 6. Review Cloud Build Pipeline**

1. In the Google Cloud Console, **Cloud Build > Dashboard**.
2. Click into **hello-cloudbuild-app** trigger to see to follow its execution and examine its logs. The last step of this pipeline pushes the new manifest to the **hello-cloudbuild-env** repository, which triggers the continuous delivery pipeline.



1. Return to the main **Dashboard**.
2. You should see a build running or having recently finished for the **hello-cloudbuild-env** repository. You can click on the build to follow its execution and examine its logs.



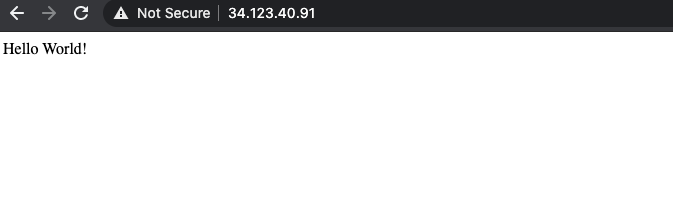
**Task 7. Test the complete pipeline**

The complete CI/CD pipeline is now configured. In this section, you test it from end to end.

1. In the Google Cloud Console, **Kubernetes Engine > Services & Ingress**.

There should be a single service called **hello-cloudbuild** in the list. It has been created by the continuous delivery build that just ran.

1. Click on the endpoint for the **hello-cloudbuild** service. You should see "Hello World!". If there is no endpoint, or if you see a load balancer error, you may have to wait a few minutes for the load balancer to be completely initialized. Click Refresh to update the page if needed.



1. In Cloud Shell, replace "Hello World" by "Hello Cloud Build", both in the application and in the unit test.

cd ~/hello-cloudbuild-app

sed -i 's/Hello World/Hello Cloud Build/g' app.py

sed -i 's/Hello World/Hello Cloud Build/g' test\_app.py

content\_copy

1. Commit and push the change to Cloud Source Repositories.

git add app.py test\_app.py

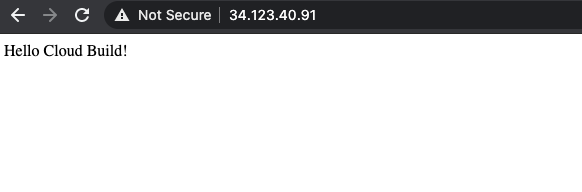
git commit -m "Hello Cloud Build"

git push google master

content\_copy

1. This triggers the full CI/CD pipeline.

After a few minutes, reload the application in your browser. You should now see "Hello Cloud Build!".



Click *Check my progress* to verify the objective.

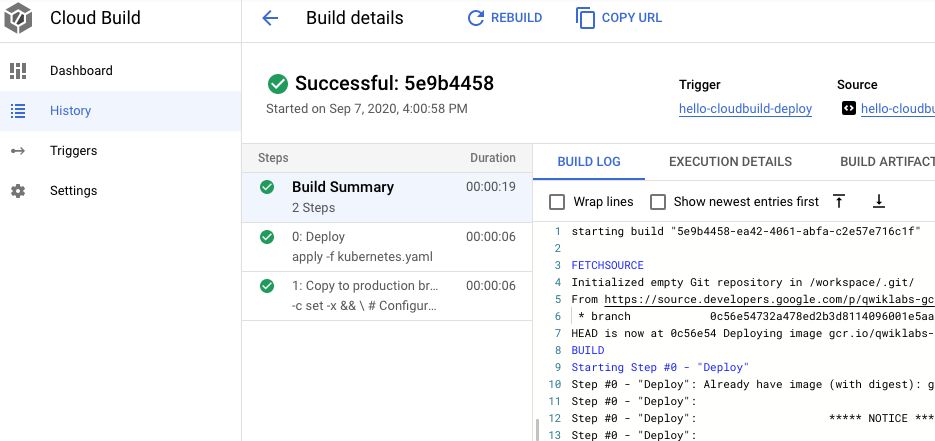
Test the complete pipeline

Check my progress

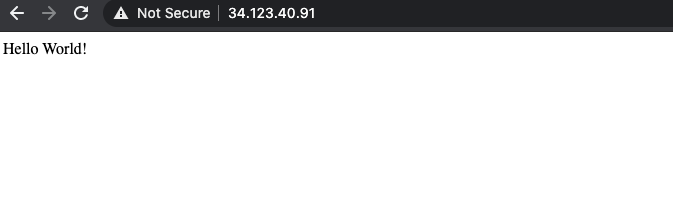
**Task 8. Test the rollback**

In this task, you rollback to the version of the application that said "Hello World!".

1. In the Google Cloud Console, **Cloud Build > Dashboard**.
2. Click on *View all* link under **Build History** for the **hello-cloudbuild-env** repository.
3. Click on the second most recent build available.
4. Click Rebuild.



When the build is finished, reload the application in your browser. You should now see "Hello World!" again.



Click *Check my progress* to verify the objective.

Test the rollback

Check my progress

