**Building Apps with GCP:**

**Cloud benefits**

- allows you to migrate all of your stored data off premises and greater redundancy for data storage

- Scalability (can remove additional resources and remove them as required),

- portability (leverage things like infrastructure as code which are heavily build on use of cloud and move around),

- cost effective (only paying for what you’re using)

Big 3:

* GCP
* AWS
* Azure

GCP benefits

* Simple and intuitive UI
* Cheaper
* Billing easy to understand
* Lower costs

IaaS services – compute engine (cloud maintains underling infrastructure and you configuring how it runs above the server itself)

PaaS – cloud SQL server, App engine, cloud storage,

SaaS end user software – gmail, google drive, google docker

You wouldn’t want to install Jenkins server on cloud instance as google would clean up instance after non use as changes are not persisted when machine is terminated

* Wouldn’t want to run any workloads that take too long to complete because will get time outs and cloud shell connection is closed
* Shell time out can be changed in settings

**Creating object in cloud shell**

**GCP organisation**

**-resources-** single compute instance, storage bucket, sql server, etc

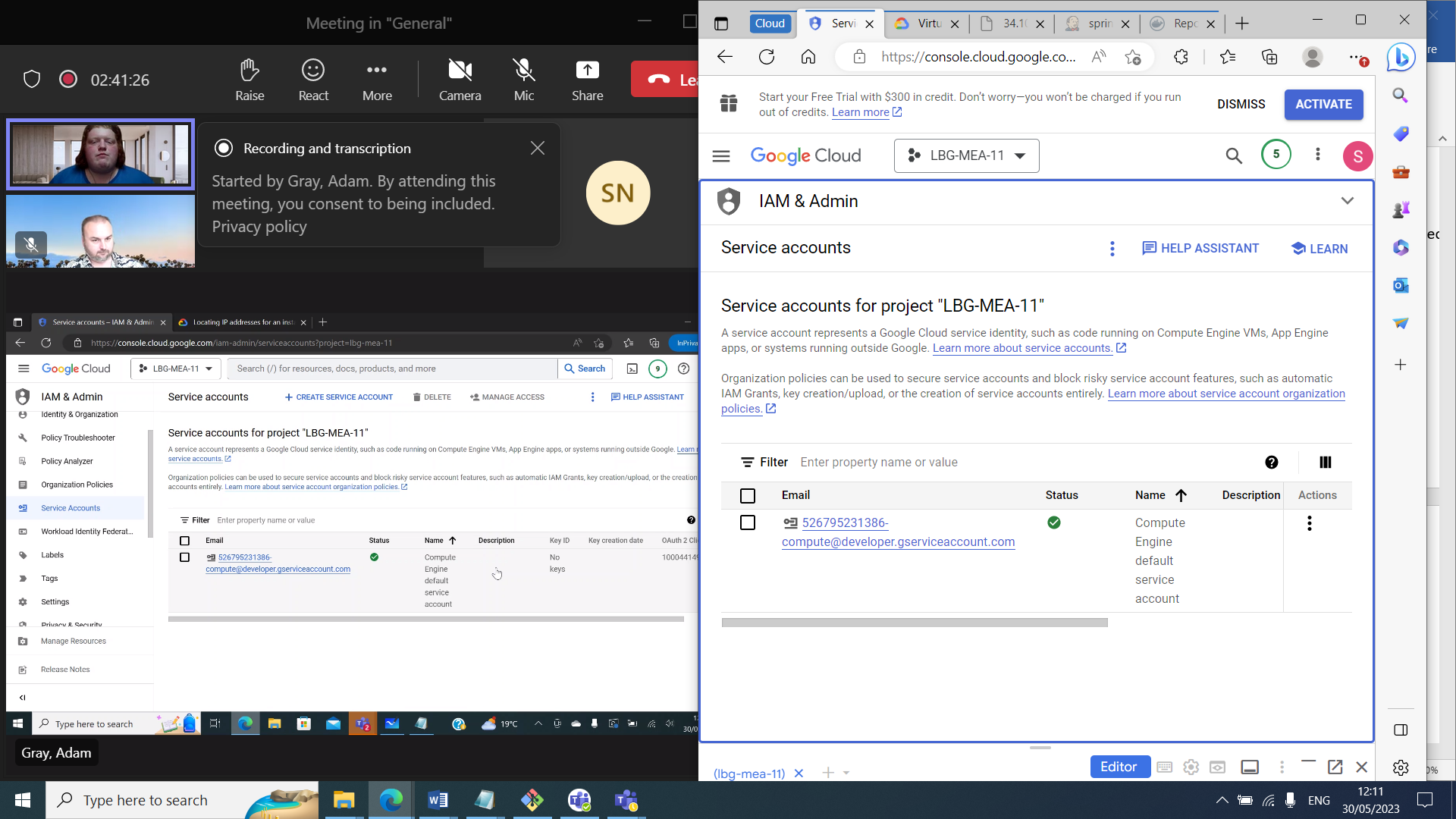
**Resources** are then grouped into **projects (**useful to view billing per project, view each project per environment with unique namespaces)

**Cloud shell has CLI pre-installed that can also be installed outside of shell too**

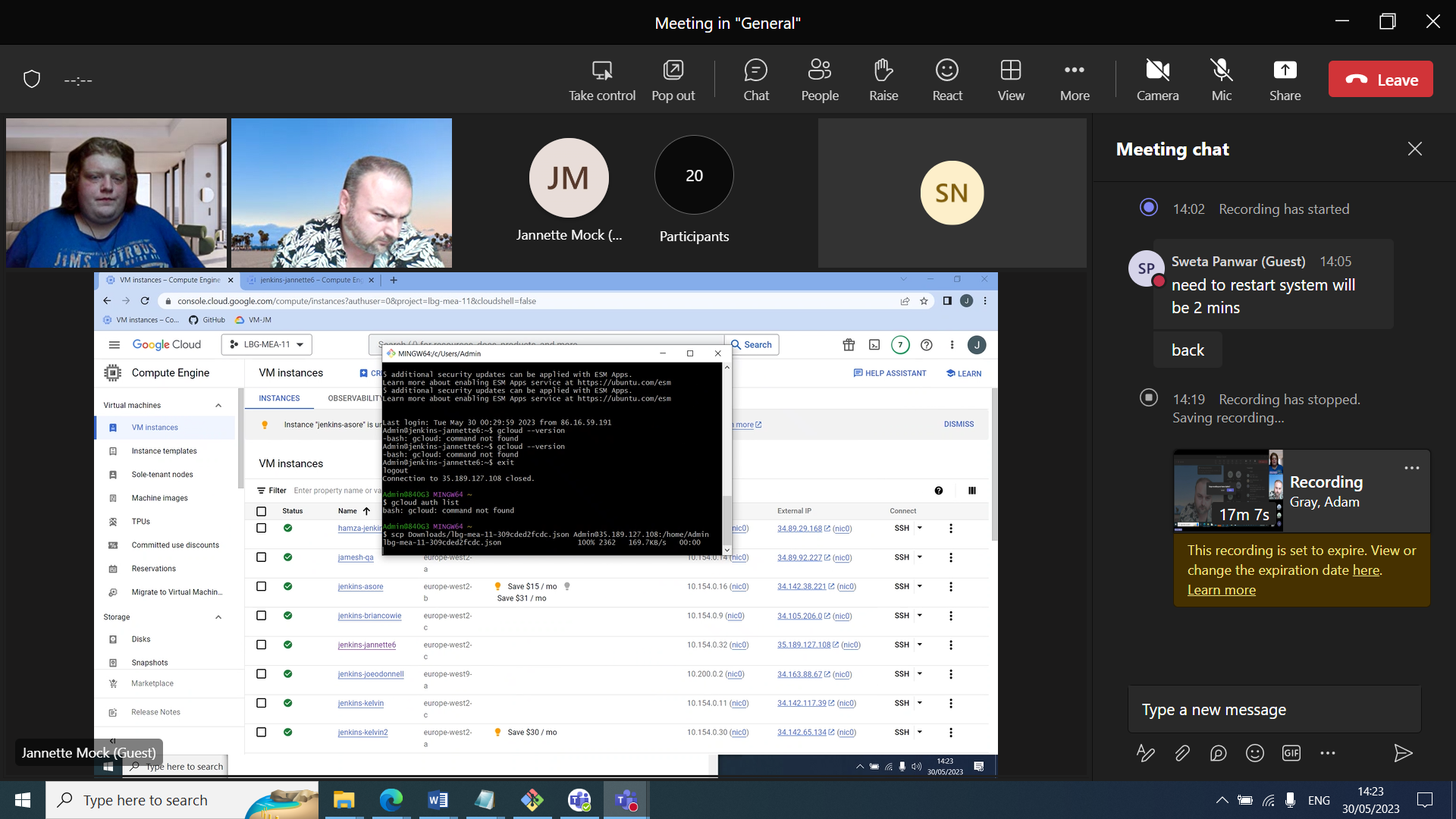
**gcloud compute instances describe jenkins-saf1 --zone europe-west2-c --format='get(networkInterfaces[0].accessConfigs[0].natIP)' -gets the ip address**

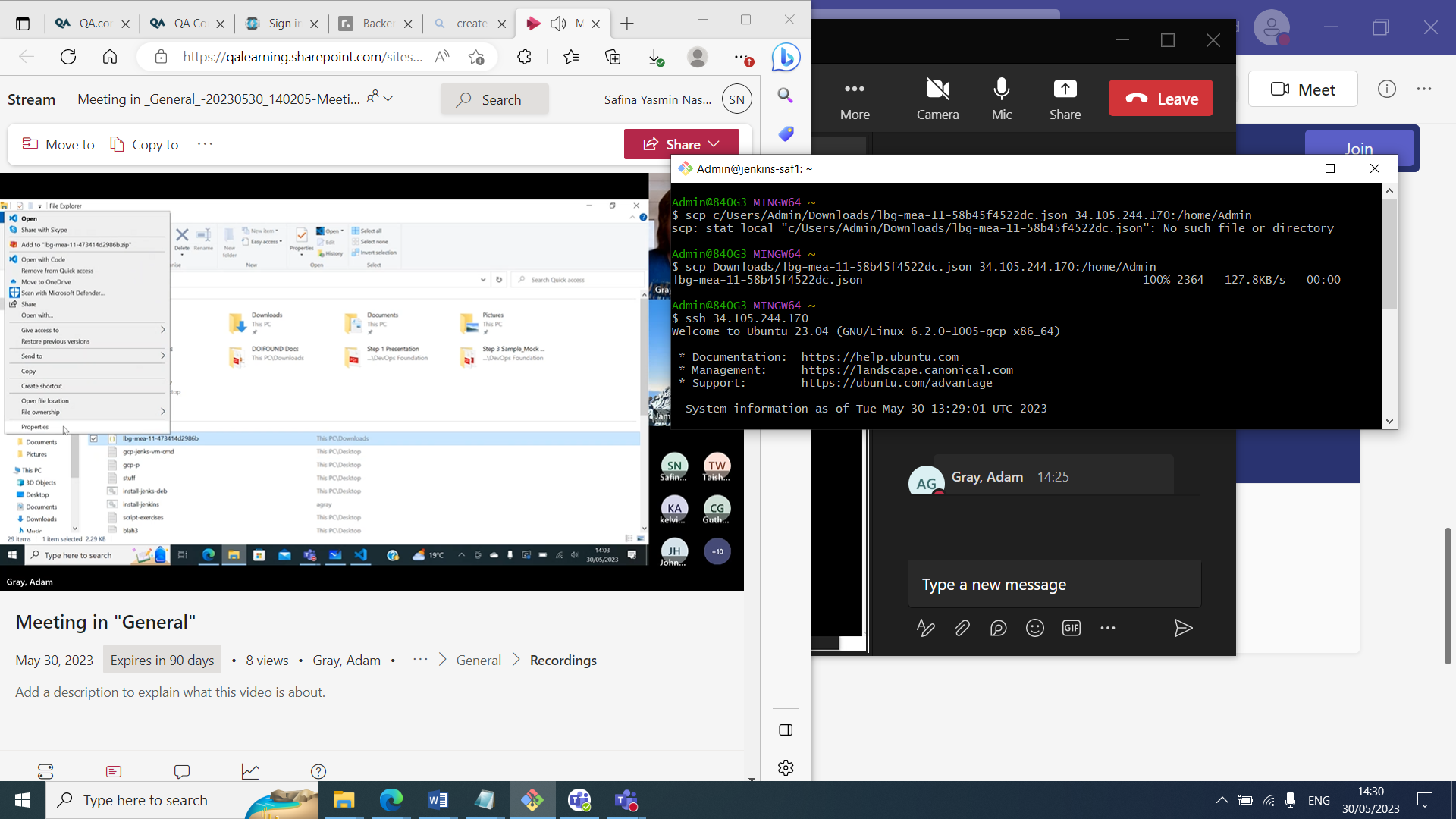
**IAM – Identity Access Management**

User accounts – used by real users service accounts



Set up service account on Jenkins server so it can utilise this service to manage storage

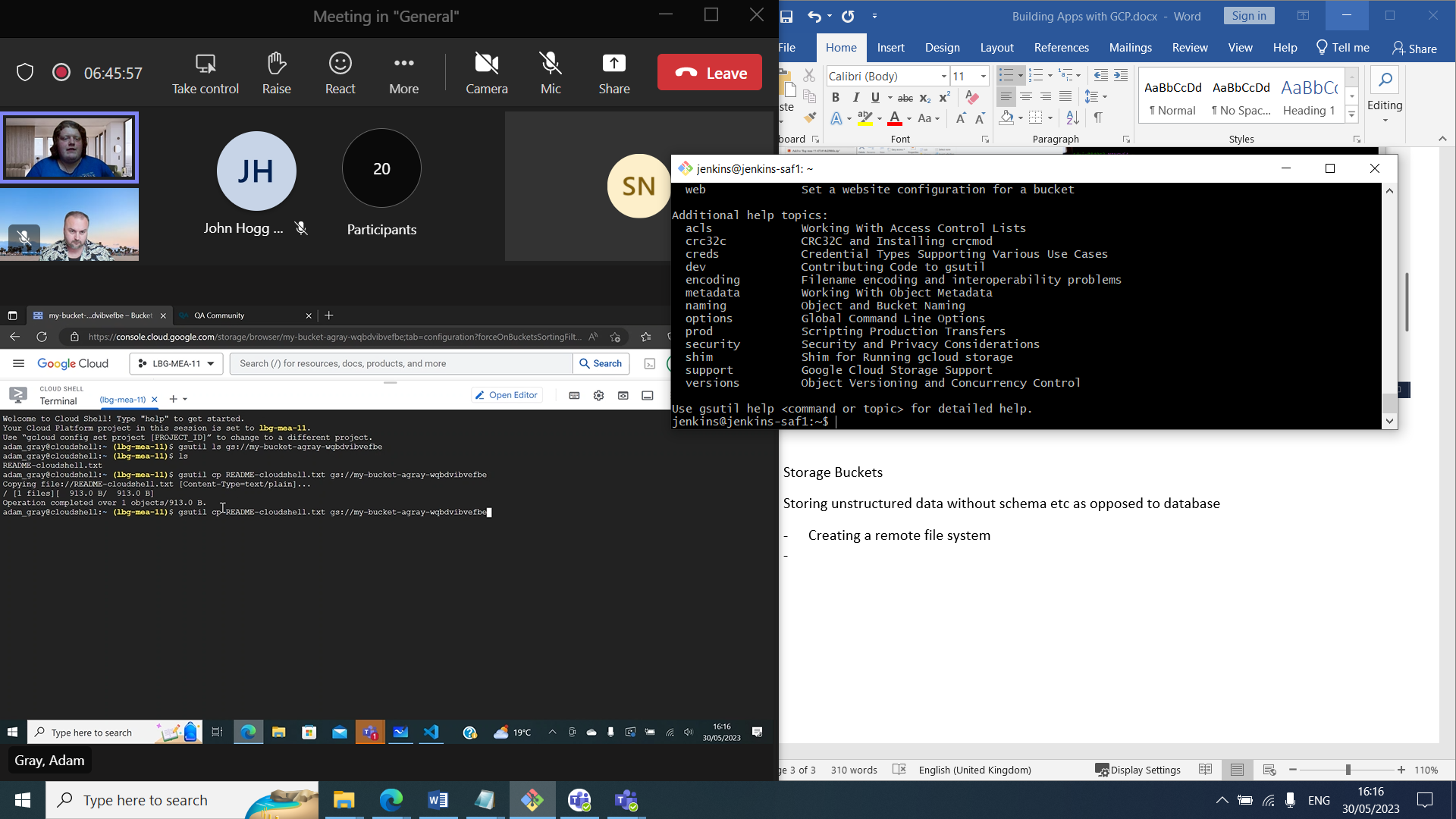


Compute engine is GCPs IaaS and VMs (isolated OSs working together that are multi tenancy),

Storage Buckets

Storing unstructured data without schema etc as opposed to database

* Creating a remote file system



**Compute services**

The most commonly used GCP compute services are:

* App Engine - Platform as a Service (PaaS) to deploy applications (including Java, Node.js, Python, .Net and Go).
* Compute Engine - Infrastructure as a Service to provision and run virtual machines.
* Google Kubernetes Engine (GKE) managed Kubernetes.
* Cloud Functions - Functions as a Service (FaaS) engine supporting many language runtimes (including Java, Node.js, Python, .Net and Go).
* Cloud Run - service dedicated for running OCI containers without Kubernetes clusters.

**Storage & Database services**

The most commonly used GCP storage and database services are:

* Cloud Storage - Object storage service for storing binary data.
* Cloud SQL - managed database services supporting MySQL, PostgreSQL and Microsoft SQL Server.
* Cloud Bigtable - Managed NoSQL columnar database service.
* Cloud Spanner - Scalable relational database service.
* Cloud Datastore / Firestore - NoSQL document database.

**Network services**

The most commonly used GCP network services are:

* VPC (Virtual Private Cloud) - Managing software-defined networks.
* Cloud Load Balancing - Software-defined, managed service for load balancing the traffic.
* Cloud Armor - managed WAP (Web Application Firewall) securing your application against DDoS attacks.
* Cloud CDN - Google Content Delivery Network.
* Cloud DNS - Distributed and highly available authoritative DNS service.

**Big data services**

The most commonly used GCP big data services are:

* BigQuery - Scalable enterprise data warehouse for SQL-based analytics.
* Cloud Dataflow - Managed Apache Beam service for stream and batch workloads.
* Cloud Dataproc - Managed Apache Hadoop and Apache Spark.
* Cloud Composer - Managed Apache Airflow service for workflow orchestration.
* Cloud Datalab - Managed Jupyter Notebook service for data analysis and visualization.

**Messaging services**

The most commonly used GCP messaging service Cloud Pub/Sub. Highly scalable messaging platform supporting both point-to-point and publish-subscribe communication models.

**Management services**

The most commonly used GCP management services are:

* Cloud Console - Web application to manage Google Cloud Platform resources.
* Cloud Shell - Browser-based Linux shell.
* Cloud IAM - Identity & Access Management (IAM) service for defining user, service account and roles.
* Operations suite (formerly Stackdriver ) - Monitoring, Operations, logging and diagnostics.
* Cloud Deployment Manager - Infrastructure As Code tool to manage Google Cloud Platform resources.

**Regions and zones**

* GCP offers data centres in over 25 regions all over the world.
* Each region is divided into multiple zones.
* Currently, there are over 80 zones available all over the world.
* Application workloads should be deployed over multiple availability zones to ensure the reliability of your services in a case of a single zone failure.

GCR Container Registry

## Overview

Google Container Registry service, or GCR for short, is a highly available Container registry service from Google Cloud Platform. GCR is a great choice if you are running your containerised applications in GCP and you need a secure and scalable registry for hosting your container images.

### Containers and API compatibility

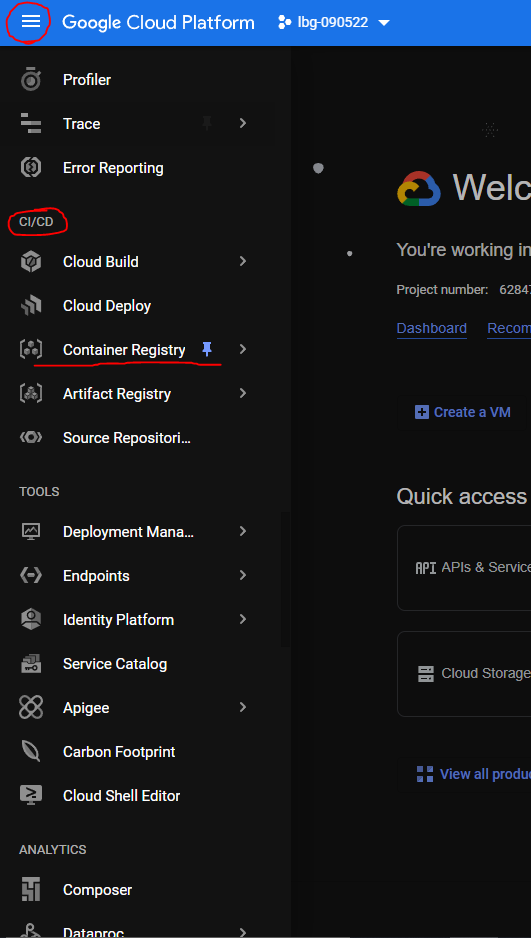
GCR service is compatible with Docker API. It supports container images in the following formats:

* Docker Image Manifest V2
* OCI

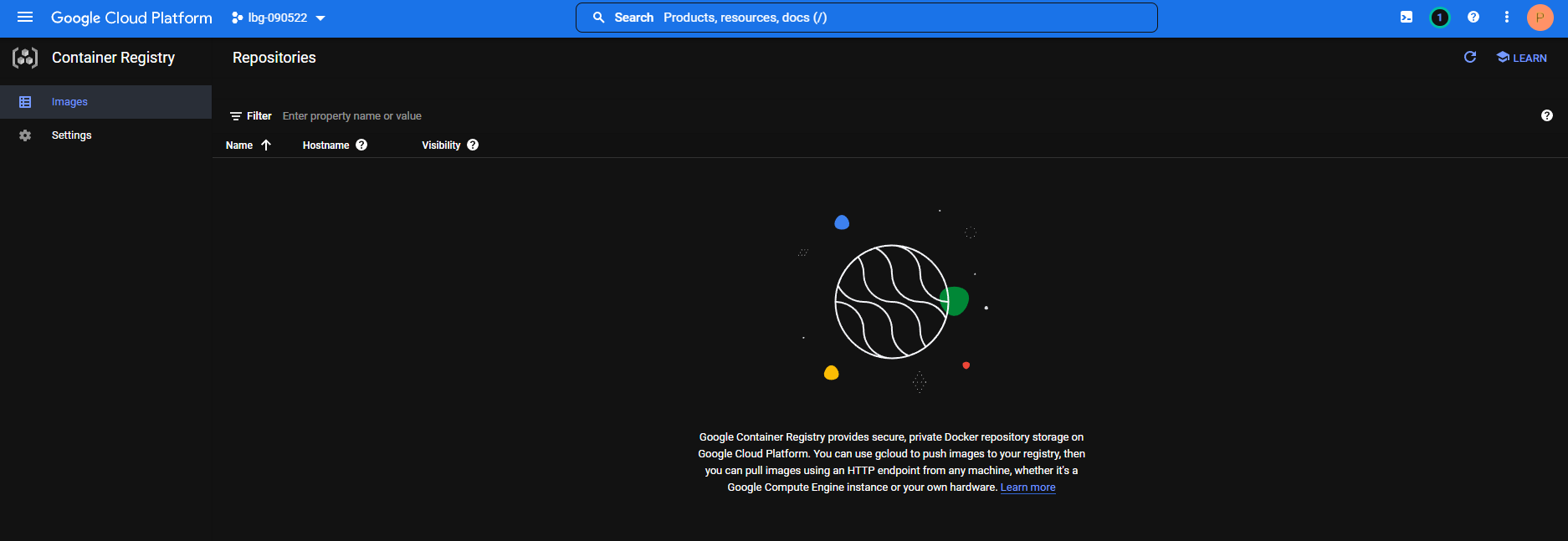
In practice, it means that any existing Docker client can access the Google Container Registry service without any issues. The experience will be very similar to using DockerHub.

## Tutorial

Let's try to demonstrate how to use an image registry from the GCP to host your container images. First of all, open the GCR service in the Google Cloud Console. You will find it in the "CI/CD" section of the services list.

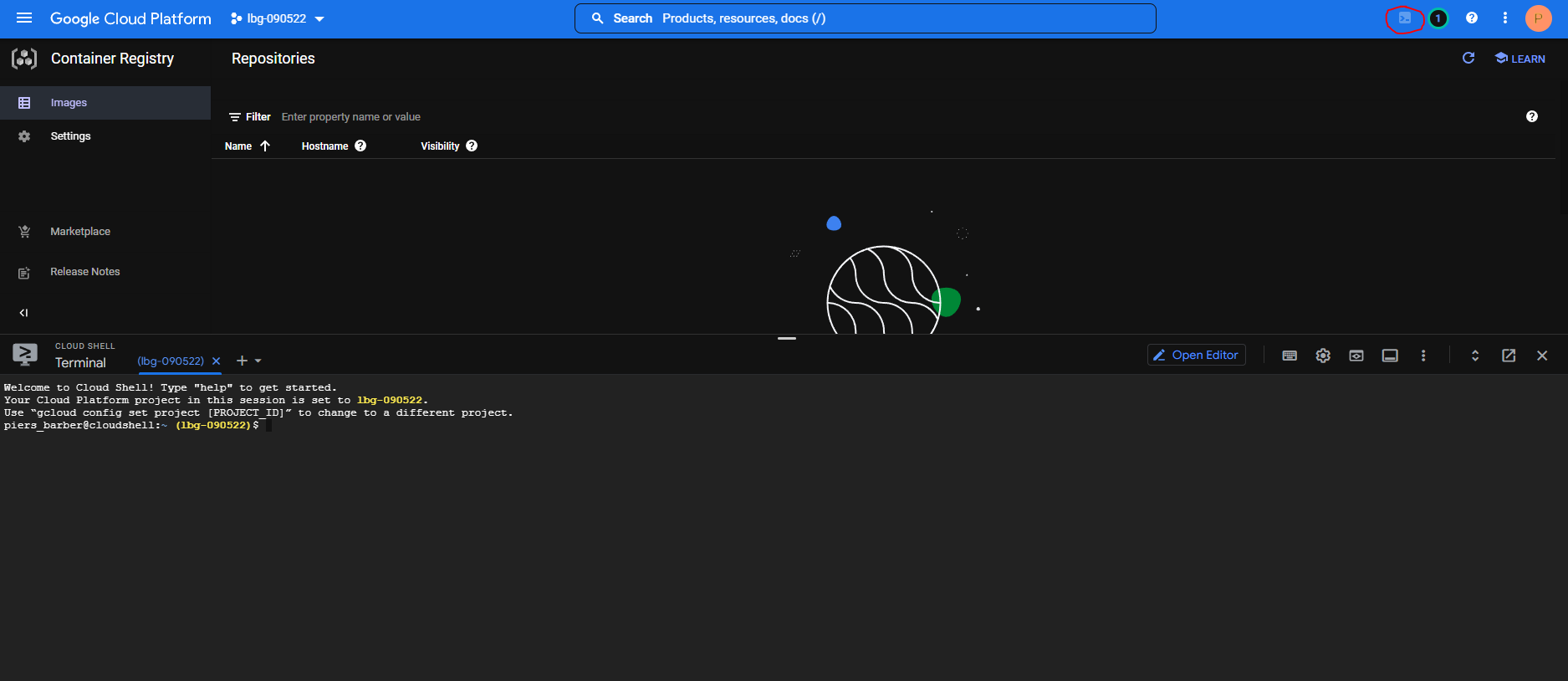


After clicking the "Container Registry" button you should see an empty registry page.



It is empty because we have not added any images there yet. Let's change it! Open Google Cloud Shell window.

You may be asked by Google Shell to authorize as soon as you execute any command in your shell. In such a case, just click the blue "Authorise" button.



Now we will create a new Docker image from a scratch and push it into GCR. Start by creating a new directory in the Google Cloud Shell.

mkdir hello

Now navigate to this new directory.

cd hello

As a next step, create a file named Dockerfile with the following content:

touch Dockerfile

nano Dockerfile

**FROM** ubuntu

**RUN** echo 'Hello world!' > /tmp/hello.txt

**ENTRYPOINT** ["cat", "/tmp/hello.txt"]

The Dockerfile above defines a container image that uses Ubuntu Linux as a base and reads Hello world! message from a file.

Now build a Docker image named hello locally, by executing the docker build . -t hello command:

docker **build** -t hello .

You can quickly verify if the container image was created successfully:

docker run -**it** hello

As you can see our container prints a welcome message "Hello World!" as we expected.

Now let's try to push our image into GCR! As a first step, you need to tag your image to indicate that it is supposed to be stored in GCR. You can do it using the following command:

docker **tag** **hello** gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1

The tag above indicates that your local hello image should live in GCR in your GCP project. We also assigned version v1 to our image.

Now execute the command docker push gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1 to push our image into GCR.

$ docker push gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1

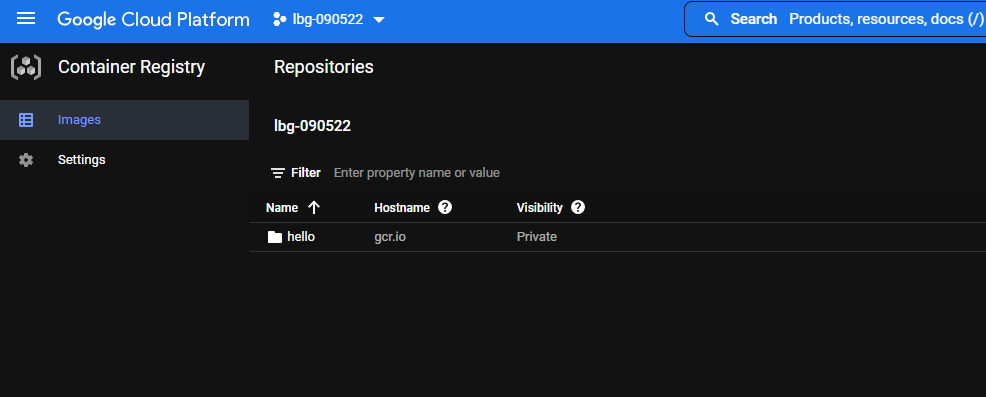
The push refers to repository [gcr.io/YOUR-GCP-PROJECT-NAME/hello]

46835fde2631: Pushed

7555a8182c42: Layer already exists

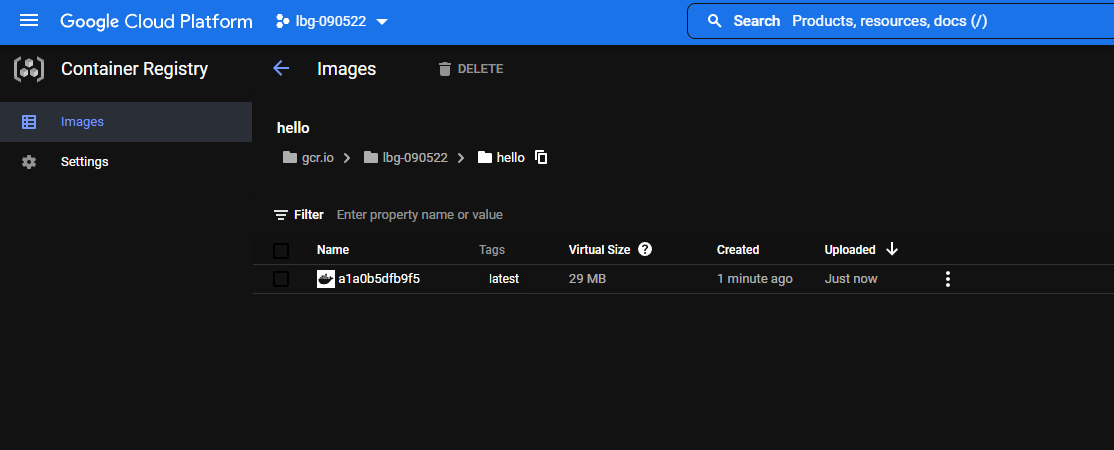
v1: digest: sha256:3c1df0a974520421ed32ae26ef45a607be329391c9ae7e1a01c28a6b649b76ee size: 736

After pushing your image, hit the "Refresh" icon in the GCR console to see the results:



As you can see GCR has a dedicated folder for our hello image now.

Inside the hello folder you can see our v1 container image we have just pushed from Google Shell.



Now imagine that your coworker would like to run this image on their local machine with Docker and gcloud installed on it. They could just try to run the docker run -it gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1 command, but they will see a message similar to the one below.

$ docker run -it gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1

Unable **to** find image 'gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1' locally

docker: Error response **from** daemon: pull access denied **for** gcr.io/YOUR-GCP-PROJECT-NAME/hello, repository does **not** exist **or** may require 'docker login': denied: Permission denied **for** "v1" **from** request "/v2/YOUR-GCP-PROJECT-NAME/hello/manifests/v1".

See 'docker run --help'.

It is because they have to authenticate to GCR before downloading the image. If they share the same GCP project with you, they can just use the gcloud auth configure-docker command to authenticate against GCR.

$ gcloud auth configure-docker

Adding credentials **for** **all** GCR repositories.

WARNING: A long list **of** credential helpers may cause delays running 'docker build'. We recommend **passing** the registry **name** **to** configure **only** the registry you are **using**.

**After** **update**, the **following** will be written **to** your Docker config file

located at [/home/hekonsek/.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)?

Docker **configuration** file updated.

After successful authentication, your coworker can execute a container created from the image you published without any issues.

$ docker run -**it** gcr.io/YOUR-GCP-PROJECT-NAME/hello:v1

Hello world!

Google Container Registry - Why Use Registries?

* Same version control and have a single source of truth and easier to store images/share with team members
* Reduces risk of losing images
* Cost effective
* Facilitates orchestration and compatible

Types of registry

* Dockerhub default registry – good for images intended for public use
* Cloud container registries
* Most control but not so scalable if self-hosted
* More configurable security, lower latency

