Before you start, please take a look at the (somewhat simplified) infrastructure overview we run at Overstory (in the next pages) so you have a bit of an understanding where certain components or services fit within the larger picture and so you also have an idea what you will be working on if hired :)

Assignment:

Part 1.
Diagram an architecture overview of how you would set up a scalable google cloud based infrastructure that can serve all these tasks efficiently within a kubernetes cluster. Keep redundancy in mind (multi-region, clustered, etc), but also costs.

Please do not spend more than **30 min** on this part, as it's meant as a way for you to learn more about Overstory's infrastructure so we can answer any questions you might have as well. A single image we can discuss together is sufficient.

Part 2
Take an already trained pytorch model and set it up to run it as a service in a kubernetes cluster. See the full assignment at
https://colab.research.google.com/drive/1osWg3AxbbyEIOgVb942JA93_vdIqpIO4?usp=sharing

Please try not to spend more than 4 hours on this part of the assignment.

Let us know if you have any questions about the assignment and good luck!

INFRA OVERVIEW:

Cloud provider and services

Overstory's infrastructure is currently running on Google Cloud Platform. The services currently used are mainly:

- Google Kubernetes Engine

  Kubernetes is used as a foundation for most of the services provided to our data scientist. This includes the Jupyterhub instance that is used by our data scientist to develop code using Jupyter notebooks from their browser and the dask clusters.

- Google Cloud Storage

  The majority of our data is stored in GCS buckets since it represents the most scalable solution for our needs.

- Google Filestore

A small amount of data (10TB) is hosted in Filestore. We are planning to reduce our usage of Filestore since it becomes expensive very fast and cannot be easily downscaled.

We also make use (on a much smaller scale) of:

- Google AppEngine
- Google Cloud SQL
- Google Cloud Run

Most of our infrastructure is described, maintained and upgraded using infrastructure as a code. In particular, we use Terraform in combination with Terraform Cloud to apply changes to our infrastructure.

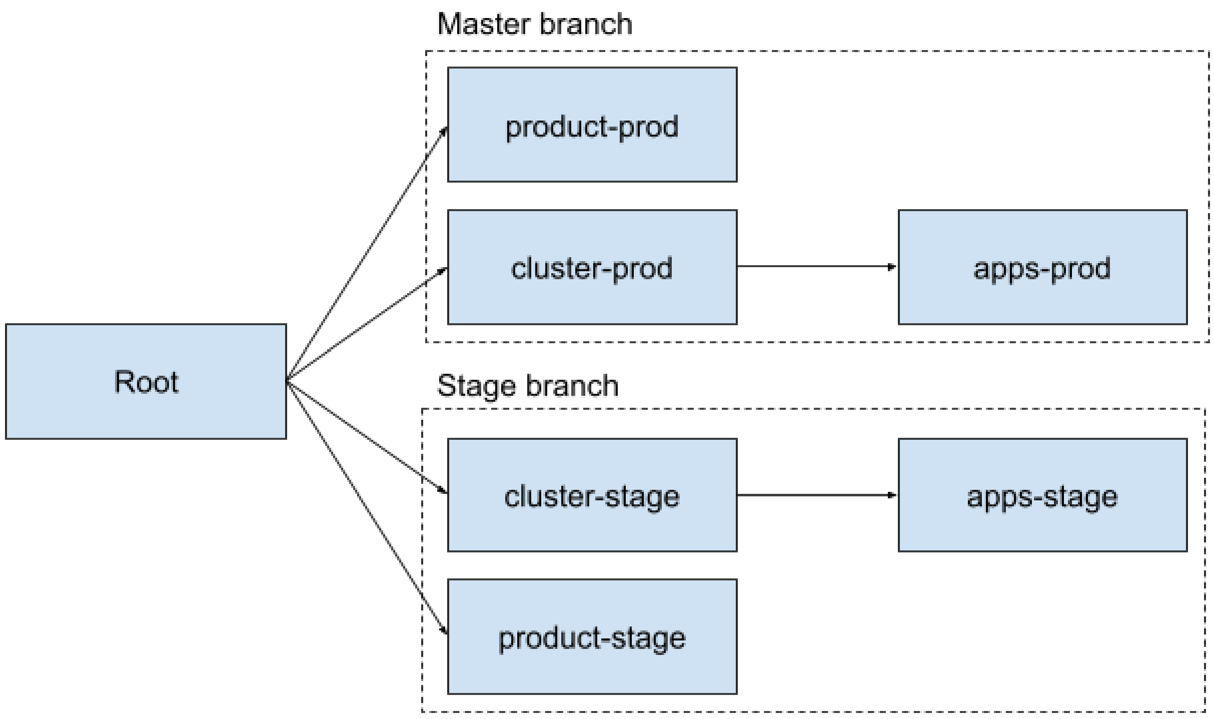
There are currently two separate environments (production and staging) living inside the same Google Project. Most of the resources are not shared across the environments and isolated (e.g. service accounts, DNS).

Code management

All infrastructure as code is stored in a GitHub repository and modified using CI/CD pipelines.

Thanks to Terraform Cloud, all the code changes in “master” are automatically deployed to production and the code in the “stage” branch is automatically deployed to the staging environment. If the master and stage branch are differing, a Github action creates a new pull request from stage to master listing the changes that are running in stage but that haven’t been deployed to production yet.
An automated system to update dependencies like terraform modules and docker images was put in place using Renovate Bot (https://github.com/renovatebot/renovate). Every Monday morning the bot scans the repository and opens one PR (to the “stage” branch) for every dependency that needs to be upgraded.

Our Terraform cloud workspaces configuration and function:



- **Root** - This workspace monitors the “root” folder of the repository on the master branch. This folder contains our Terraform cloud configuration as code and generates all the following workspaces with their configuration and dependencies.
- **cluster-prod / cluster-stage** - These workspaces monitor the “elm-cluster” folder of the repository, respectively on the master and on the stage branch. This folder contains the cluster configuration and every resource that is linked to the cluster. E.g. GKE, node pools, DNS, Google Service accounts, etc.
- **apps-prod / apps-stage** - These workspaces monitor the “elm-apps” folder of the repository, respectively on the master and on the stage branch. This folder contains the applications that are deployed in the GKE cluster.
- **product-prod / product-stage** - These workspaces monitor the “app” folder of the repository, respectively on the master and on the stage branch. This folder contains the configuration of our product, mostly based on Cloud Run and Identity Aware Proxy.

Kubernetes management

Kubernetes is running a mix of regular and preemptible instances, large computations are run mostly on preemptible machines. Every data scientist works on a single, not shared, regular instance that is selected from a menu and runs a Jupyter notebook and (optionally) a virtual desktop environment.

The Kubernetes cluster is configured as a private cluster where all nodes are private and not exposed to the internet. Authentication is required for all endpoints and is performed using Google OAuth.

Deployments on Kubernetes are managed using Helm and the terraform helm provider (link).

Processing large amounts of data is done using Dask (https://dask.org/) running on preemptible instances. Data scientists can request a dask cluster from their jupyter notebook and submit work to it. The cluster is automatically scaled up and down based on the computation needs.

Another more recent addition on how we process data is using Dagster (after trying Argo Workflows and other Airflow-like variants). Dagster runs in Kubernetes and creates a separate Pod for every running pipeline. Data scientists can develop pipelines locally and deploy them by pushing to master for staging and by tagging a new version for production.

Google Cloud Storage Management

Most of our data is stored in Google cloud storage. We use a one bucket per client model, as well as some general buckets. We found GCS to be the most price efficient storage solution for our needs. Moreover, blob storage allows us to transfer large volumes of data quickly even with concurrent workloads.

Data is pulled and pushed to Google cloud storage using mainly their python library (link).

Machine Learning

Machine learning tasks are initially developed using jupyter notebooks and committed as CI-able libraries using nbdev (see https://www.overstory.com/blog/how-nbdev-helps-us-structure-our-data-science-workflow-in-jupyter-notebooks), and when taken to production wrapped into Dagster using the dagster specific operation model.
We use Data parallelism to train on multiple GPUs on a single machine at the same time, and use Distributed Data Parallelism on multiple machines with multiple machines for the largest training jobs.

Inference is run adhoc for our data products whereas no "real" realtime inference is required by our customers as of yet (this will likely change in the future). For very large inference jobs (nation scale on high resolution satellite data), we have our own python based pipeline for ingesting large amounts of satellite imagery, predicting on it, and aggregating and converting predictions into geospatial file formats.

Product

The Overstory Explorer is what our customers use to view the data products we have created for them and is run as a serverless React App for which we have a stage and prod environment
The web frontend requests data (vector and raster data: think “google maps” for trees) over an API service build on top of FastAPI.
FastAPI in turn serves satellite imagery through our Titiler server, and vector data through Martin.