Markdown Converter Infrastructure Design

Cloud Run Services

The frontend and backend API is handled by Cloud Run Services. Each container serves the frontend using a basic python Flask application, and handles `/api/` requests that come in from the client.

API Endpoints:

- `/api/mdconvert` handles POST requests containing markdown from the client that needs to be converted and creates a pubsub message with the final GCS bucket that the converted HTML should be placed in. Once we publish a message to pubsub, we then return the `/api/retrieve/<filename>` endpoint that the client should poll against until the file is done. The filename will be the pubsub message ID appended to `.html`.
- `/api/retrieve/<filename>` handles GET requests from the client waiting for a converted file. Returns 404 if the file is not located on the GCS mountpoint. This is expected until the file is converted and uploaded to the GCS bucket.
- `/health` health check endpoint for Cloud Run to determine container status.

GCS Mounts:

- Each tenant (aka customer) has a Cloud Run Service per environment (stage, prod). They also have a GCS bucket that their converted files will be uploaded to and retrieved from. Each Cloud Run Service will only have IAM access to their appropriate bucket and have that mounted at `/mnt/bucket` for the API to easily retrieve converted files.

Pub/Sub

Each environment (stage, prod) has a pub/sub topic. When messages are published they will be consumed by the appropriate Cloud Function for the environment.

Cloud Functions

Each environment (stage, prod) has a Cloud Function. This function's purpose is to take messages from the appropriate Pub/Sub topic and convert the markdown in the message data

into html. Then the output html is uploaded to the appropriate GCS bucket defined by the message attribute `bucketname` using the message ID in the format `<messageId>.html`

GCS

Each tenant (aka customer) has a GCS bucket for each environment (stage, prod) that their converted html files will be stored in.

Design Considerations

Choice of Compute:

- Cloud Run Services can easily handle basic API interactions with the client and can easily scale by environment considerations.
- Cloud Functions do the “heavy load” of converting our markdown to html and can scale independently of the Cloud Run Services. This also helps prevent any attack vector from the API if we had “secret sauce” happening in the conversion.
- GCS buckets allow us to store our converted artifacts and can easily be stored or cleaned up over time. They could also be another easy way to serve out the converted files to customers if they didn’t want to use the frontend client.

Handling Multi-Tenancy:

- Our terraform and infrastructure is designed in such a way that it can easily be scaled to any number of clients. Each tenant defined in the terraform variable will get created a Cloud Run Service and GCS bucket for each environment (stage, prod). As well as each Cloud Run Service only have access to the appropriate GCS bucket via proper IAM service accounts with least privilege.

Ingestion Strategy:

- Pub/Sub allows us to easily send markdown conversion requests and have them trigger a conversion via Cloud Functions.

Scaling/Resiliency:

- All services are setup to easily scale based on parameters. As well as all resources are created within a full Google Cloud region for resiliency.

Monitoring:

- We have a custom logging metric setup to track 500 errors from each Cloud Run Service. As well as an alerting strategy on that. We also have another metric setup to track if any Cloud Run Services have reached their `max_instances` setting. Allowing

us to determine if services need to be provisioned more. These metrics can be viewed via the Cloud Monitoring Dashboard we have setup.

Cost trade offs:

- As of now, costs are minimal. Everything is set to scale down to zero if not used. Though this is designed via our terraform to be scaled up based on environment if needed.