DORIS Image Viewer and Processor Application Design

Architecture Diagram Walkthrough

The application is accessible on the public internet and the process begins at the web client and browser level. The client first connects to the external load balancer and firewall, which is managed by DoITT, either through HTTP (port 80) or HTTPS (port 443). HTTP connections are only used for redirecting to HTTPS connections and this is encouraged by the federal government (https://https.cio.gov/#footnote-3). Traffic is then forwarded, through HTTPS only, from the external load balancer to the load balancer for the application. The load balancer then sends the traffic to the app server for the listening Nginx service to process. Nginx also serves as an external cache for all of the static files (html, css, js) and allows the application to be more efficient. Nginx proxies the remaining traffic (non-static endpoint) to the Python WSGI app. The application receives the request and returns a response. The Python app can connect to the PostgreSQL database to read/write records through Consul and its ingress gateway. The application uses the cloud provider API, strictly through HTTPS, to transfer and retrieve files from cloud storage. NYC.ID is an external service for user authentication in the application and is accessed through HTTPS only.

Security

In securing the overall environment, only the necessary ports are opened for all the resources. Port 22 for SSH, port 443 for HTTPS, ports in the 2100-21255 range for consul, and port 5432 for the PostgreSQL database. Additionally, password authentication is disabled and private key authentication is used for server authentication.

Data is secure in transit with the use of both consul and the HTTPS protocol. Consul is used to secure service-to-service communication over mutual TLS. Communication of data between the Python App and the PostgreSQL database is secure through the consul connect proxy. Files transferred to and from the object storage and the python application are secured through requiring HTTPS for all requests.

In securing file uploads, staff authentication and authorization checks should be in place in the application to verify permissions. Additionally, all files are virus scanned on upload using uvscan and proper MIME type is validated. Celery is also used for efficiency and allows for asynchronous uploads.

Data is secure at rest by disabling password authentication for all servers, preventing SSH outside of Citynet and securing resource parameters. Securing the environment innately provides security of the database and objects at rest. Configuring resource parameters such as requiring HTTPS for the storage API, setting proper data classification levels, and enabling cloud security tools for recommendations and advisory are some further steps in securing data at rest.

External Cache and Asynchronous Tasks

As previously mentioned in the architecture diagram walkthrough, Nginx is used to serve application static files and maintains a cache. Celery and Redis are used in the python application to handle asynchronous tasks and maintain cache respectively. Virus scanning on file upload and transferring files to cloud storage are asynchronous tasks to be handled by Celery and Redis.

High Availability

The use of DevOps tools, an external load balancer, an application load balancer, Nginx service, and cloud scaling resources allows for the application to be highly available. Using DevOps tools such as terraform and ansible allows for the efficient creation and configuration of resources. If a resource needs to be redeployed, it can be completed quickly with the use of terraform and ansible. The external load balancer and firewall managed by DoITT serves as the first layer of defense. The application load balancer, such as Azure Load Balancer, is highly performant and is capable of handling millions of requests per second while maintaining high availability. Nginx on the app server also provides high availability by serving application static files, setting rate limits, and configuring whitelists. Leveraging cloud scaling resources such as Azure virtual machine scale sets and Amazon EC2 Container Service are integral to a highly available application. The cloud scaling resource is configured using an image of the application, which can be created using Docker, and provides redundancy and improved performance. Load is shared among the virtual machines in a scale set for availability and efficiency. Clients can still access the application even if a virtual machine is down as another virtual machine in the set will be used to handle requests.

Image Post Processing

Utilizing a cloud provider's CDN service, such as Azure Content Delivery Network and Amazon CloudFront is a secure and efficient method for image post processing. Functionality such as scaling, cropping, rotating, sharpening, blurring, pipeline, watermark brightness and contrast, format conversion are all readily available in Azure Content Delivery Network. The image rendering is achieved through query strings on the created Azure CDN URL. This solution is ideal if the intended post processing is standardized across images (scale image by %, add watermark, etc).

If post processing needs to occur on an image by image basis, code changes in the application as necessary. A frontend interface will be developed to allow staff to edit and process images. Once the staff has finished processing an image, the edited image will be transferred to cloud storage and its location is stored in a column in the database. There will be separate database columns for the original image and the edited image.