Orca: Cloud Data Orchestration

The Problem

Running complex data pipelines locally is often unsustainable due to limited computational power and scalability, prompting individuals and companies to turn to the cloud. However, automating cloud resources is not without its challenges. It demands time, expertise, and significant learning, often leading to delays and increased costs. This inefficiency is so prevalent that many companies dedicate entire teams or employees to manage these tasks. Without effective solutions, businesses risk bottlenecks in their workflows, reduced productivity, and higher operational expenses, highlighting the critical need for streamlined and automated cloud management systems to save time, reduce costs, and improve efficiency.

The Solution

Orca provides a streamlined solution by automating the linking, execution, and management of cloud resources. It eliminates manual overhead, prevents bottlenecks, and ensures reliable task execution through features like SQS-driven order enforcement. By saving time, reducing costs, and simplifying cloud workflows, Orca empowers users to focus on their core objectives.

Features

- **Automated Resource Linking**
    - Simplifies the process of connecting cloud resources (e.g., EC2 instances, S3 buckets) using IAM roles and credentials, reducing setup time and complexity.
- **Order Enforcement with SQS**

    - Utilizes Amazon SQS to maintain execution order and prevent race conditions in running pipelines, ensuring consistency and reliability.
- **Centralized Pipeline Management**
    - Enables users to easily add, remove, run, or stop pipelines, with output stored in S3 buckets.
- **Cloud-to-Cloud Communication**
    - Facilitates seamless data exchange between the Orca Cloud and User Cloud, ensuring efficient task execution across environments.
- **Reliability and Efficiency**
    - Guarantees smooth execution of workflows through Orca's infrastructure, minimizing errors and downtime.

To Do:

Front-End

- [x] ~~Login Page~~
- [x] ~~Register Page~~
- [ ] Task Summary Page
- [ ] Task Overview Page
- [ ] Script Upload Page
- [ ] Task Scheduling Page
- [ ] Cloud Setup Page

Back-End

- [x] ~~User Authentication Handler~~
- [x] ~~Cloud Link Handler~~
- [ ] Upload Script Handler (Directory Architecture has to be determined)
- [ ] Task Execution Handler (orca-side)

- [ ] Get Task Info
- [ ] Task Scheduler

Cloud Infrastructure

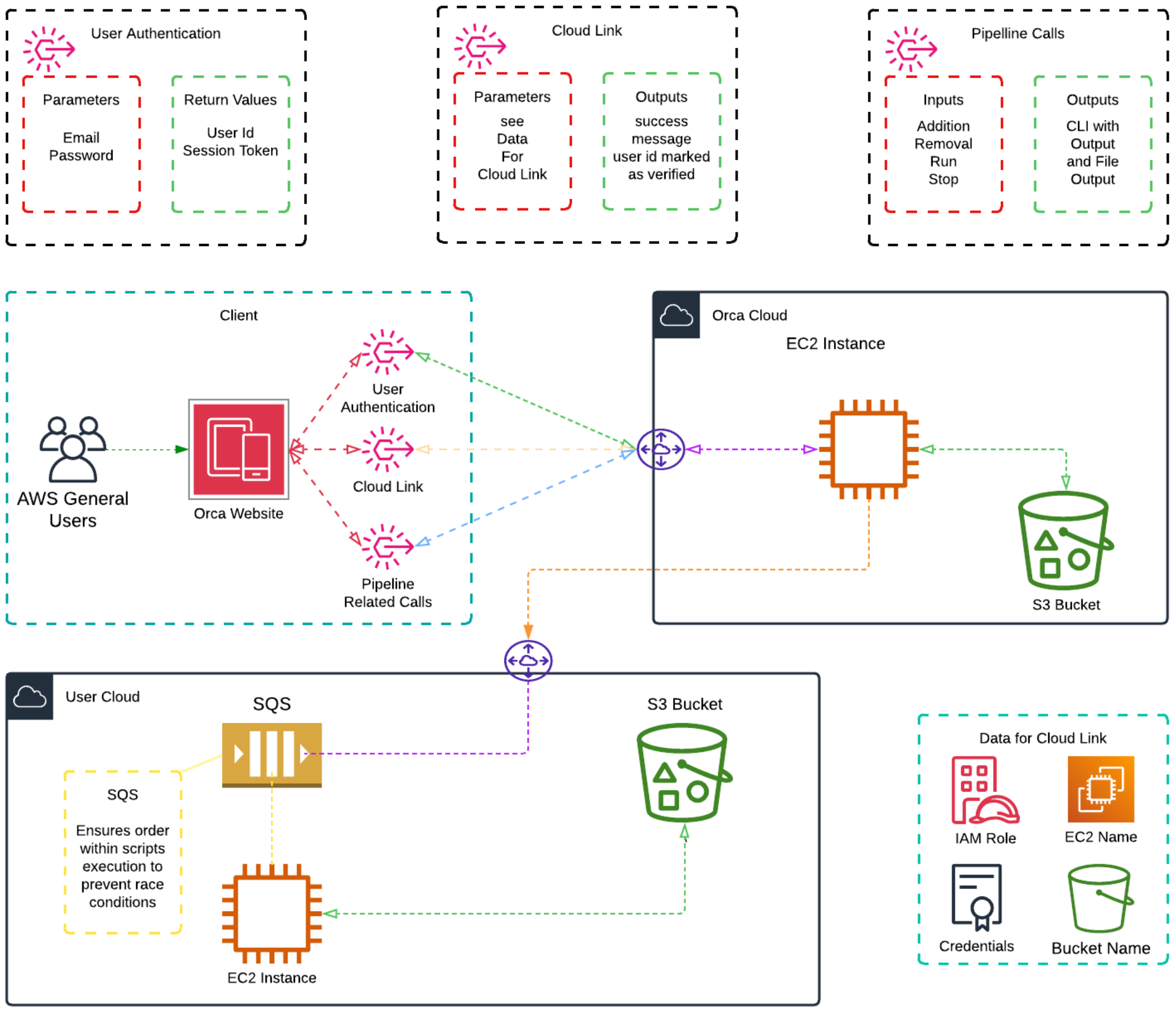
- [x] ~~Create orca EC2~~
- [x] ~~Set up IAM policy and Role~~
- [x] ~~Install SSM and ensure Back-End can communicate with EC2~~
- [x] ~~Set Up Nginx Webserver in EC2~~
- [x] ~~Link domain name to EC2 IP~~
- [x] ~~Set up SSL cert for https~~
- [ ] Deploy Back-End in EC2
- [ ] Create test EC2
- [ ] Ensure communication works between both EC2s
- [ ] Create a different cloud account and create documentation for set-up
- [ ] Test EC2 on test-user cloud account

Documentation

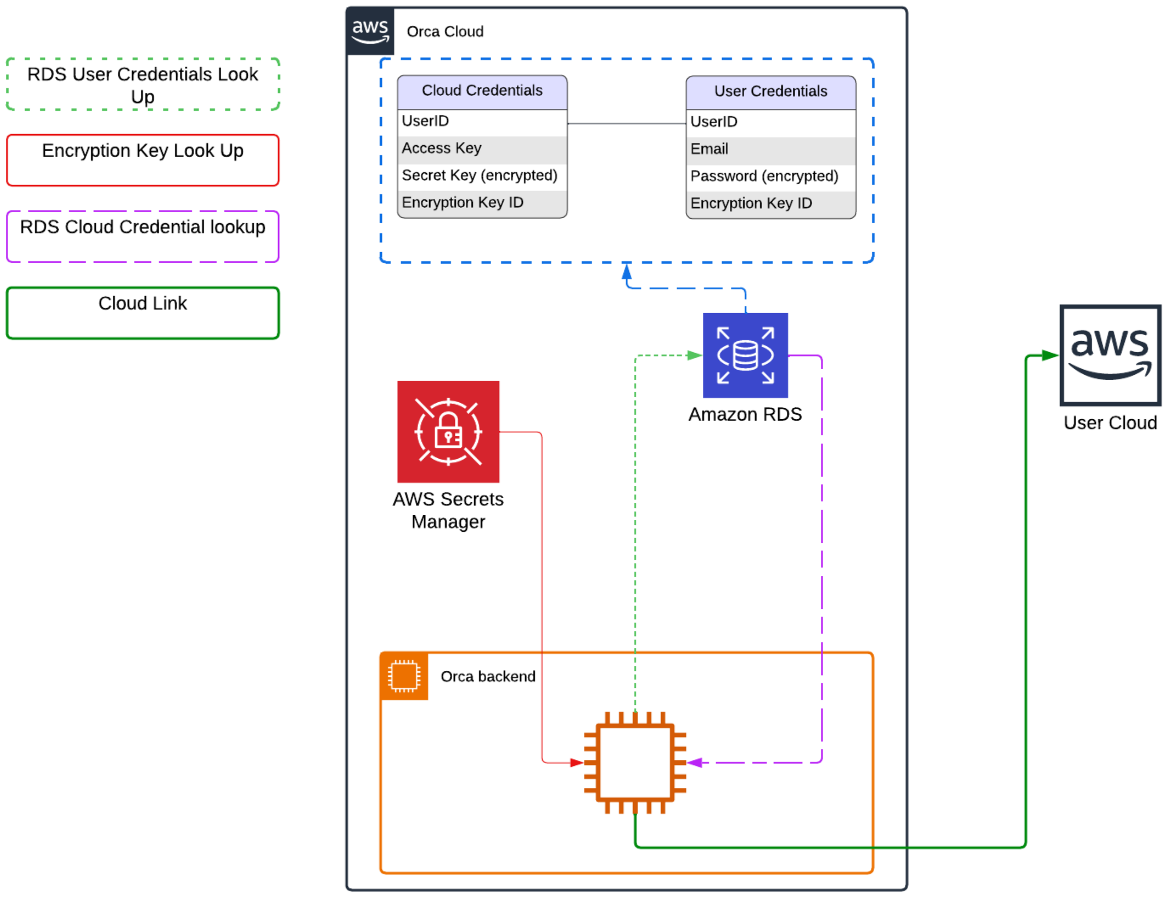
- [x] ~~Define problem solution~~
- [x] ~~Define features to prevent scope creep~~
- [x] ~~Create architecture diagram~~
- [x] ~~Develop To Do dashboard using notion~~
- [ ] Create documentation for each component and research required
- [ ] Write a potential and current blockers section
- [ ] Develop detailed documentation for cloud setup page
- [ ] Sources

Architecture

Main Architecture



Cloud Link API Call



Components

Component	Technology/Service	Features
Frontend	Vue.js	- User-friendly interface - Upload files - Schedule tasks - View results - Secure user authentication
Backend	Python (Flask)	- Handles uploads - Stores files - Executes tasks securely - Manages task scheduling
Cloud Hosting	AWS	- Reliable cloud infrastructure

Storage	S3	- Stores uploaded files and task outputs - Directory based
Task Execution	EC2	- Runs tasks in isolated, secure environments
Database	RDS	- Tracks Login Credentials - Tracks Cloud Credentials
Authentication	IAM Role	- Ensures secure access - Provides a simple setup guide for users
Task Management	Boto3	- Concurrent task execution - Automated scheduling

Component Research and Analysis

Front-End

The frontend of Orca is developed using Vue.js and will be hosted on Nginx running on the Orca EC2 instance. Nginx was chosen over Apache due to its lightweight, resource-efficient nature and its event-driven architecture, which ensures lower resource consumption—making it an ideal fit for this project. Additionally, I am familiar with Nginx, which further streamlined the setup process. To enhance security, CertBot was used to obtain an SSL certificate, ensuring the website is served over HTTPS. Firewalld was configured to enforce HTTPS traffic, adding another layer of protection. Fail2Ban was installed as a precautionary measure to mitigate brute-force attacks and unauthorized access attempts. This configuration prioritizes performance, security, and reliability.

Back-End

The backend of Orca is built using Flask and is hosted on an EC2 instance for reliability. The application is served through Gunicorn, which efficiently manages multiple worker processes for handling requests. Nginx is configured as a reverse proxy, enhancing security and performance by managing SSL termination and load balancing. The backend connects to a MySQL database, with the connection managed through boto3 to ensure seamless integration with AWS services. All sensitive credentials and configurations are securely stored and accessed via AWS Secrets Manager, providing encryption at rest and controlled acces

Cloud Handler

Orca’s Cloud Handler is developed using python. It utilizes the boto3 AWS SDK library to communicate with both EC2 and S3 resources

Works Cited

1. https://medium.com/@Anita-ihuman/hosting-a-static-website-with-aws-ec2-using-nginx-4cbc0424ea41
2.