Plaitful Systems Architectural Design Document

with ManaBurn as a showcase project

V 0.1

MoroLuckyDragon

Feb 2024

Table of Contents

Intended Audience:

Architects, Developers, Engineers

Purpose:

To communicate architecture design with a plan to scale, roadmaps.
Start with REST architecture and transitioning to Microservices architecture for scale.

Initial Design Focus and Priorities:

- **Clear Domain Boundaries**
  Start by defining clear domain boundaries for each of your components (Database API, Encounter API, Quest API, User Profile API, Image Engine API, Narrative API). Each should encapsulate a specific domain of functionality. This is crucial for minimizing work during the transition to microservices since each of these components can eventually evolve into its own microservice.
- **Design RESTful Endpoints**
  For each component, design RESTful endpoints that adhere to REST principles, such as using HTTP methods appropriately:

  GET for fetching data,
  POST for creating data,
  PUT/PATCH for updates,
  DELETE for deletions)
  and structuring URLs to reflect resource hierarchies.

  Encounter API might have endpoints like **/encounters/{id}**, **/encounters/{id}/details**, etc.
  Quest API could include **/quests/{id}**, **/quests/{id}/challenges**, etc.
  User Profile API might encompass **/users/{id}**, **/users/{id}/profiles**, etc.
  Image Engine API could offer **/images/{id}**, **/images/search**, etc.
  Narrative API might provide **/narratives/{id}**, **/narratives/{id}/elements**, etc.
- **Implement Business Logic Separation**
  Ensure that the business logic for each domain is well encapsulated within its service. This means that the Encounter API, for example, should not directly manipulate data belonging to the Quest API. Instead, it should call the Quest API's endpoints if it needs information or actions related to quests. This separation helps in transitioning to microservices by ensuring that each service is already operating independently.
- **Database Considerations**
- Each component should ideally interact with its own database schema or even a separate database, depending on your constraints. This will greatly simplify the process of moving to a microservices architecture, where each service manages its own data.
- **Documentation and Contracts**
  Documented your API endpoints and their expected inputs/outputs rigorously. This documentation will serve as a contract between different parts of your application and is invaluable when services begin to operate more independently as microservices.
- **Transition to Microservices**
  When you're ready to transition to a microservices architecture, you can start by:
  1. **Decoupling Deployment**: Begin deploying each component independently. This might involve containerization (using Docker) and orchestration tools (like Kubernetes) to manage deployments and scaling.
  2. **Dedicated Databases**: Ensure each service has its own database if you haven't done so already.

3. **Implement Service Discovery**: As you break down your architecture into more granular services, implementing a service discovery mechanism will become crucial for services to dynamically discover and communicate with each other.
4. **Introduce an API Gateway (eventually)**: While you're starting without an API Gateway, introducing one as you scale and transition to microservices will help manage entry points, aggregate responses, and handle cross-cutting concerns like authentication, logging, and SSL termination.

Systems Architecture

The diagram is divided into three main layers: **Client Layer**, **API Gateway Layer** (optional for now but planned for future scalability), and **Service Layer**. Below the Service Layer, you would have the **Data Storage Layer** and the **Infrastructure Management** section.

1. **Client Layer**:
    - This includes different types of clients like web browsers, mobile apps, or external systems that interact with your APIs.
2. **API Gateway Layer** (Optional/Future Use):
    - An API Gateway acts as a single entry point for all client requests, routing them to the appropriate microservice. This layer can also handle cross-cutting concerns such as authentication, rate limiting, and analytics.
3. **Service Layer**:
    - **Database API**: Manages direct interactions with the database(s) and provides database-related functionalities to other services.
    - **Encounter API**: Handles game encounters, communicating with the Database API as needed.
    - **Quest API**: Manages quest-related functionalities, possibly interacting with the Encounter API and Database API.
    - **User Profile API**: Deals with user data, preferences, profiles, etc., interacting with the Database API for storage and retrieval.
    - **Image Engine API**: Responsible for image processing, storage, and retrieval, could use external storage services or the Database API for metadata storage.
    - **Narrative API**: Manages storytelling elements, narratives, etc., possibly requiring interaction with other APIs for comprehensive data management.
4. **Data Storage Layer**:
    - Databases (relational or NoSQL) that each service interacts with. Ideally, each microservice would have its own database to ensure loose coupling and service autonomy.
5. **Infrastructure Management**:
    - Demonstrates the use of IaC tools like Terraform, AWS CloudFormation, or Ansible for provisioning and managing the AWS EC2 instances, networking, databases, and other required infrastructure. This could also include container orchestration tools like Kubernetes (EKS) or ECS for managing Docker containers.

2024 Technology Roadmap

Current Implementation/scope:

- Closed LLM integration with OpenAI's GPT-4-turbo
- Input themes and concepts from veteran game creatives
- Unity frontend demo for Android
- IAM setup for backend services
- Process and tools to enable creatives to interact with Plaitful system
- Cloud infrastructure for the following:
  - Generative AI work completed:
    - Analysis and comparison of Mistral, Llama, Starling, BetaLlama LLMs for out of the box narrative generation
    - Research into fine tuning open-source and closed-source LLMs, langchains, prompt/token optimization to reduce content generation costs per unit
    - Image AI Engine: EC2 GPUs for fast processing of image content via a finely tuned stable diffusion engine
    - Narrative Engine: EC2 instances in AWS cloud for narrative content generation
  - Database and Caching infrastructure for each API set up
  - Devops deployments (image, encounter and prompt) via Docker for all APIs

March 2024:

- Deploy image ai engine on cloud
- Deploy auto cache deployment plugin in generation workflow.
- Flush out and POC narrative generation: templatize for any onboarding project, look into langchains and fine tuning of models
- Review AWS architecture for cost efficiency (what do we need to host, how can we do it for cheap while building user base)

April 2024:

- Flush out avatar/character requirementsfor each user, user flows on how to create, what happens when an avatar is deleted, died, abandoned, transfer rules
- Refactor and finish up service/data layer (with exception of user profile portion)
- Optimize deployment flows for all APIs – one click deploys, deployment scripts via CLI

May 2024:

- Polish tooling UI and process for creatives to interact with system
- Flush out user UX flows: authentication, billing, login, session management

June 2024:

- Flush out user UX flows: with inventory, marketing or target offers (ads), crafting
- Flush out crafting, inventory, combat systems with designers

July 2024:

- Model out data models and flows, review existing architectural plan and optimize based on flushed out requirements.
- Implement user authentication, User profile API, User Profile Database

Systems Design

Current Implementation

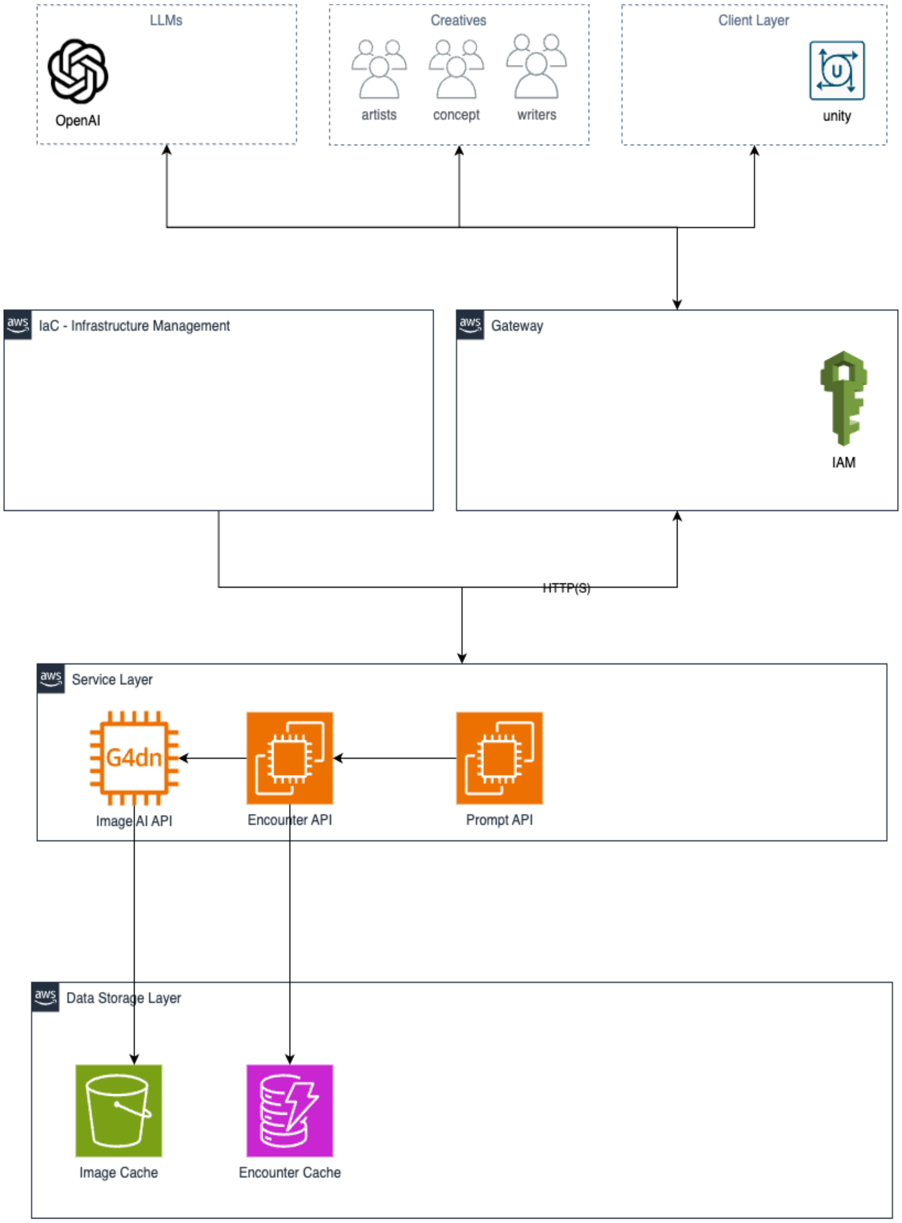


Figure 1: Developing Architecture (Feb 2024)

Targeted Implementation

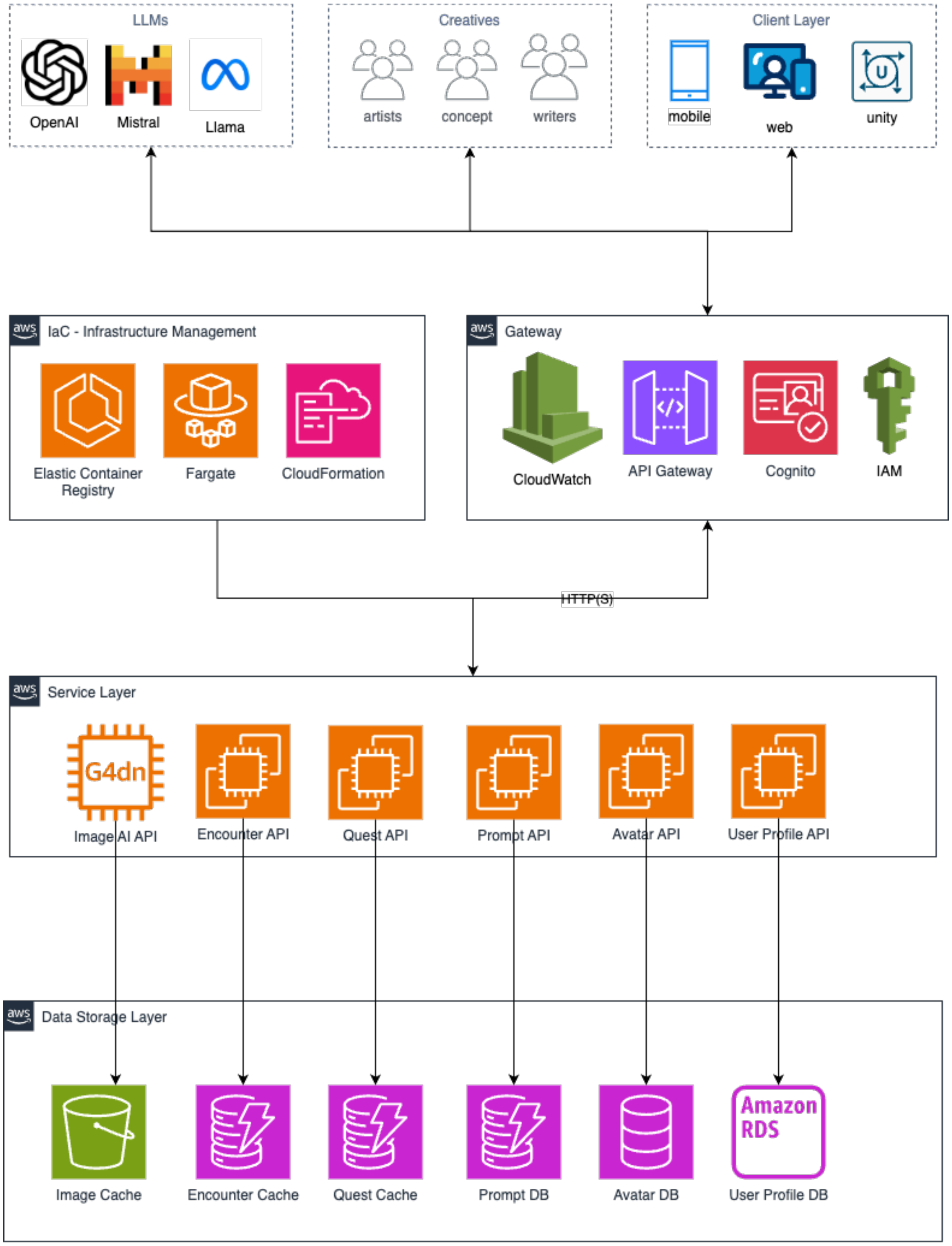


Figure 2 – Full Scalability

API Gateway Implementation

Design and Use Case Considerations

Authentication

AWS API Gateway supports several mechanisms for controlling and managing access to your APIs:

- **IAM Permissions**: Use AWS Identity and Access Management (IAM) to create roles and policies that define who can access your API and what actions they can perform.
- **Lambda Authorizers**: Custom authorization logic can be implemented using AWS Lambda functions. This allows you to authenticate tokens (such as JWT or OAuth tokens) against your own user management systems or third-party identity providers.
- **Cognito User Pools**: Integrate with Amazon Cognito User Pools to manage user identities and authentication. This is a powerful option for mobile and web applications, enabling features like user registration, authentication, and access control.

Rate Limiting

API Gateway provides rate limiting features to help protect your backend services from traffic spikes and to maintain the quality of your service:

- **Steady-State Rate Limits** and **Burst Capacity**: You can set the rate limit (requests per second) and burst capacity (the maximum rate of requests allowed over a short period) on a per-method basis in your API Gateway settings. This helps prevent your backend from being overwhelmed by too many requests.
- **Usage Plans and API Keys**: Create usage plans to specify who can access your APIs and at what rates. You can require API keys for your APIs and associate them with usage plans to enforce these limits. This is useful for managing access for different types of consumers.

Analytics

API Gateway is integrated with Amazon CloudWatch, providing detailed analytics and logging capabilities:

- **Execution Logs**: Log requests and responses to debug and monitor the operational health of your APIs. You can set the level of logging detail and retain logs for analysis.
- **Access Logging**: Capture detailed information about who accessed your API and how they used it. This can include requester IP, request headers, request paths, and more.
- **CloudWatch Metrics**: Automatically monitor performance metrics such as the number of successful and failed requests, latency, and data transfer sizes. You can set alarms on these metrics to get notified of potential issues.
- **CloudWatch Dashboards**: Create custom dashboards in CloudWatch to visualize API usage and performance data, helping you understand how your APIs are being used and how they are performing over time.

Using these AWS services in conjunction with API Gateway allows you to secure, manage, and analyze your APIs effectively, ensuring they are reliable, secure, and scalable.

IaC – Infrastructure Management

Design and Use Case Considerations

Orchestrating Microservices with Kubernetes or ECS

1. **Kubernetes**: An open-source system for automating deployment, scaling, and management of containerized applications. AWS offers Amazon EKS (Elastic Kubernetes Service), which makes it easy to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes.
2. **Amazon ECS (Elastic Container Service)**: A fully managed container orchestration service that's deeply integrated with AWS. ECS supports Docker containers and allows you to easily run, stop, and manage containers on a cluster. ECS can use AWS Fargate to remove the need to manage servers or clusters of Amazon EC2 instances.

Auto-Scaling

- **Kubernetes Horizontal Pod Autoscaler (HPA)**: Automatically scales the number of pods in a replication controller, deployment, replica set, or stateful set based on observed CPU utilization (or, with custom metrics support, on other application-provided metrics).
- **ECS Service Auto Scaling**: Automatically adjusts the desired count of tasks in your service based on CloudWatch alarms (e.g., CPU utilization or custom metrics).

Monitoring and Management

- **Amazon CloudWatch**: Provides monitoring and management for AWS cloud resources and the applications you run on AWS. You can use CloudWatch to collect and track metrics, collect and monitor log files, and set alarms. CloudWatch Container Insights is specifically designed for monitoring containers and microservices.
- **AWS X-Ray**: Helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.

Deploying APIs with Attached Instances

- **Infrastructure as Code (IaC)** tools like AWS CloudFormation or Terraform can be used to define and deploy your microservices infrastructure along with your EC2 instances, networking, and other required AWS resources as a unit. These tools allow you to script the setup of all necessary resources, ensuring that each microservice can be deployed, scaled, and managed independently.
- **Docker Compose on Amazon ECS**: For simpler use cases or development environments, Docker Compose support on Amazon ECS allows you to define and run multi-container Docker applications. With a Compose file, you can configure your application's services, networks, and volumes, and then use AWS CLI v2 integration for ECS to deploy your entire stack as a unit on ECS.

Steps to Scale Microservices

1. **Containerize** your application, if not already done, and push your Docker images to a container registry (e.g., Amazon ECR).
2. **Choose an orchestration service** (EKS or ECS/Fargate) and define your service configurations, including CPU and memory requirements, desired count, and scaling policies.
3. **Implement IaC** for provisioning infrastructure and deploying your services, ensuring that each microservice can be independently deployed and scaled.
4. **Set up monitoring and logging** using CloudWatch and X-Ray for visibility into your microservices' performance and health.
5. **Define auto-scaling policies** based on metrics that accurately reflect your workload's needs.

By following these steps and utilizing these AWS services, you can effectively deploy, scale, and monitor your microservices architecture on EC2 using Docker, handling potentially hundreds of microservices efficiently.

Features of Amazon ECR

- **Fully Managed Service**: Amazon ECR eliminates the need to operate your own container repositories or worry about scaling the underlying infrastructure. It handles the storage and management of your container images in a secure, scalable, and reliable manner.
- **Security**: ECR integrates with AWS Identity and Access Management (IAM), allowing you to specify who can access your container images. It also offers scanning capabilities that automatically scan images for vulnerabilities, helping to improve the security of your applications.
- **High Availability**: ECR is designed to be highly available and durable, storing images in multiple geographic regions and replicating them across multiple Availability Zones to increase the availability of your application.
- **Scalability**: It scales automatically to handle large volumes of requests without the need for manual intervention, making it suitable for applications of any scale.
- **Integration with AWS Services**: ECR works seamlessly with ECS and AWS Fargate, allowing for easy deployment of containerized applications. It also integrates with AWS CodeBuild and AWS CodePipeline for a continuous integration and continuous delivery (CI/CD) workflow.

Benefits of Using Amazon ECR

- **Simplified Deployment Process**: By using ECR in conjunction with other AWS services, you can streamline the process of deploying, managing, and scaling containerized applications.
- **Reduced Costs**: ECR helps to optimize costs by eliminating the need to provision and manage infrastructure for your container registry. You pay only for the amount of data you store in the registry and the data transferred to the Internet.
- **Improved Security**: With the security controls and vulnerability scanning capabilities offered by ECR, you can ensure that your container images are secure and free of known vulnerabilities.

- **Global Accessibility**: ECR supports cross-region replication of images, making it easier to manage global deployments and ensure that images are available where your applications are running.

How to Use Amazon ECR

To use Amazon ECR, you first need to create a repository in the ECR console or via the AWS CLI. Once the repository is created, you can push and pull Docker images using the Docker CLI or other compatible tools. ECR provides a secure, scalable, and reliable way to store and share container images, facilitating easier deployments and operations for containerized applications within the AWS ecosystem.

Key Concepts of IaC

- **Automation**: IaC automates the provisioning of infrastructure, reducing the potential for human error and increasing efficiency.
- **Consistency**: By defining infrastructure through code, IaC ensures consistency across development, testing, and production environments. This minimizes "works on my machine" problems and increases reliability in deployments.
- **Version Control**: Infrastructure code can be versioned and stored in source control repositories, allowing teams to track changes over time, revert to previous states, and understand the evolution of their infrastructure landscape.
- **Reusability**: Components of the infrastructure defined as code can be reused across different environments, projects, or even organizations, saving time and reducing the potential for errors.

Tools for Infrastructure as Code

Several tools facilitate the practice of IaC, with varying approaches such as declarative (what the desired state should be) versus imperative (how to achieve that state) methodologies. Popular IaC tools include:

- **Terraform**: An open-source tool created by HashiCorp that uses a declarative configuration language to manage and provision infrastructure across a variety of service providers (cloud, on-premises, or hybrid environments).
- **AWS CloudFormation**: A service provided by Amazon Web Services that allows users to define a collection of related AWS and third-party resources, provision them quickly and consistently, and manage them throughout their lifecycle.
- **Ansible**: An open-source tool that provides simple but powerful automation for cross-platform computer support, focusing on both orchestration and application deployment.
- **Puppet and Chef**: Both tools allow users to automate the configuration and management of servers and software, using code to automate the installation and configuration of software components on servers.

Benefits of IaC

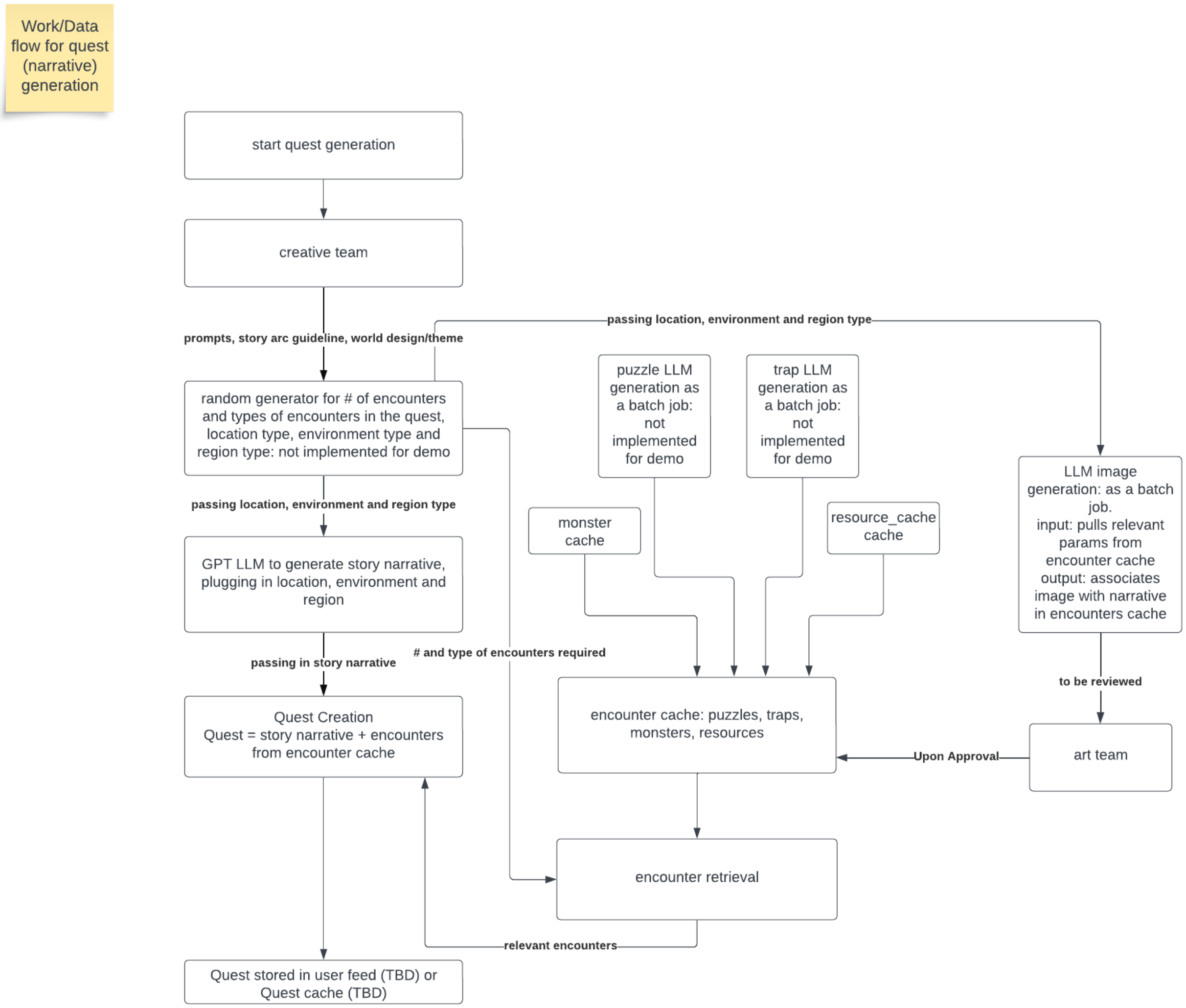
- **Speed and Efficiency**: Rapidly provision and tear down environments and infrastructure as needed, without waiting for manual processes or bespoke configuration.
- **Cost Reduction**: Automating infrastructure provisioning reduces the need for manual effort and can help in optimizing resource usage, leading to cost savings.
- **Risk Mitigation**: Consistent environments reduce the risk of production issues caused by configuration drift or discrepancies between environments.

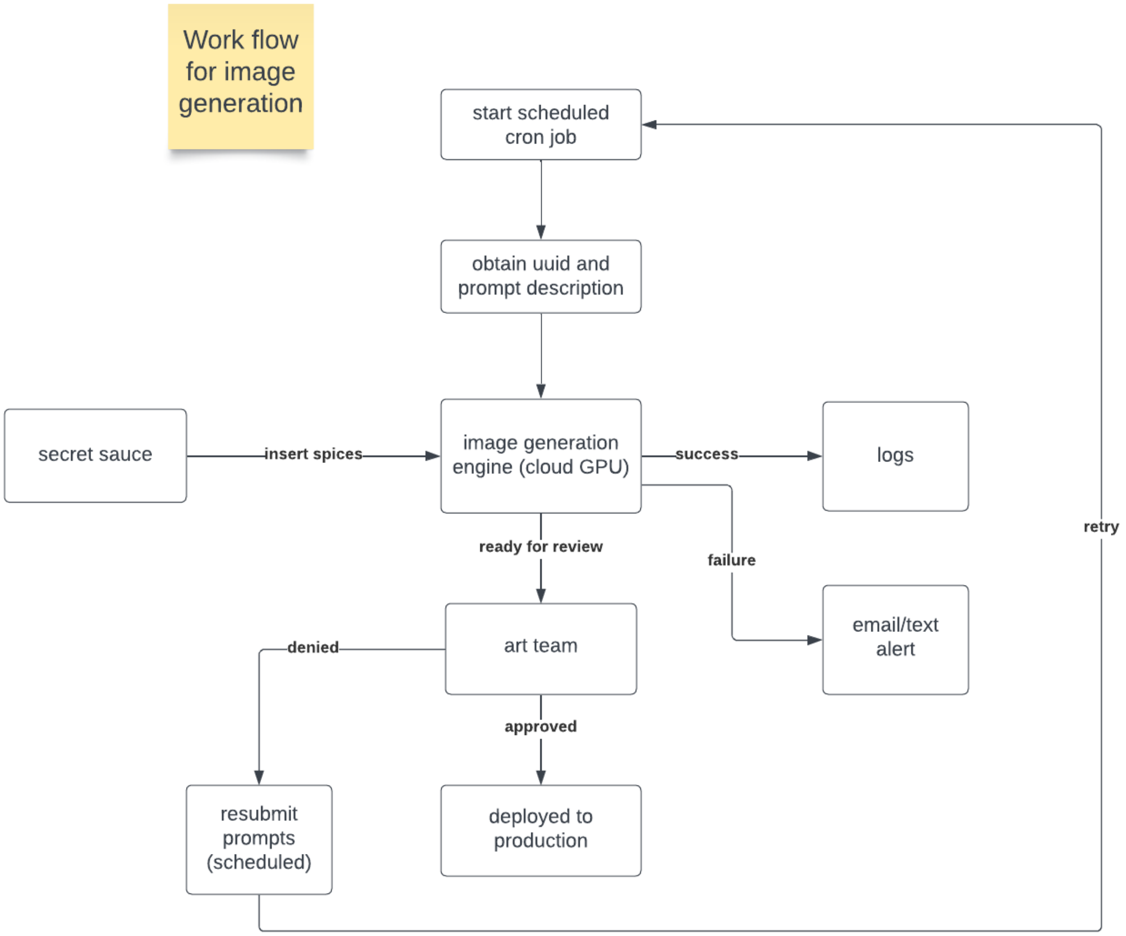
- **Compliance and Governance**: With the entire infrastructure defined as code, it's easier to enforce compliance policies and audit infrastructure changes.

Incorporating IaC into your development and operations processes can significantly improve the speed, efficiency, and reliability of infrastructure provisioning and management, making it an essential practice in modern cloud computing and DevOps methodologies.

Data Flows

Mockup and description of data flows for each system portion go here
Data transfer protocols, design and design decisions go here


Work/Data flow for quest (narrative) generation
start quest generation
creative team
prompts, story arc guideline, world design/theme
passing location, environment and region type
random generator for # of encounters and types of encounters in the quest, location type, environment type and region type: not implemented for demo
puzzle LLM generation as a batch job: not implemented for demo
trap LLM generation as a batch job: not implemented for demo
LLM image generation: as a batch job. input: pulls relevant params from encounter cache output: associates image with narrative in encounters cache
passing location, environment and region type
monster cache
resource_cache cache
GPT LLM to generate story narrative, plugging in location, environment and region
passing in story narrative
# and type of encounters required
to be reviewed
Quest Creation Quest = story narrative + encounters from encounter cache
encounter cache: puzzles, traps, monsters, resources
Upon Approval
art team
encounter retrieval
relevant encounters
Quest stored in user feed (TBD) or Quest cache (TBD)


Work flow for image generation
start scheduled cron job
obtain uuid and prompt description
secret sauce
insert spices
image generation engine (cloud GPU)
success
logs
failure
email/text alert
retry
ready for review
art team
denied
approved
resubmit prompts (scheduled)
deployed to production

User Flows

Mock of user flow scenarios go here

Generative AI

Leveraging ManaBurn as an example project

Narrative Engine

Pick your LLM

Imagine a tool to determine which LLM to use, what prompts or agents to configure to achieve a narrative that strikes the right tone with the concept you have in mind.

Prompting

Write and update prompts for your narrative

Testing Web Tool

Test your prompts with LLMs in our online test tool to determine to bring to life the concept you have in mind

Image Engine

Leveraging Stable Diffusion and a series of tweaks, fine tuning of parameters, we can obtain a consistency in theming and feel across the board for (any) concept.

Refer to Figure 5 above under "Data Flows" to see how we envision this process.

We will design a tool that encompasses the following:

Web UI for Creatives

Everyone can develop their own "secret sauce" via a web UI tool.
A Cloud-based web UI tool for creatives to tweak and set up the generator to produce images that is consistent in theme based on particular concepts.

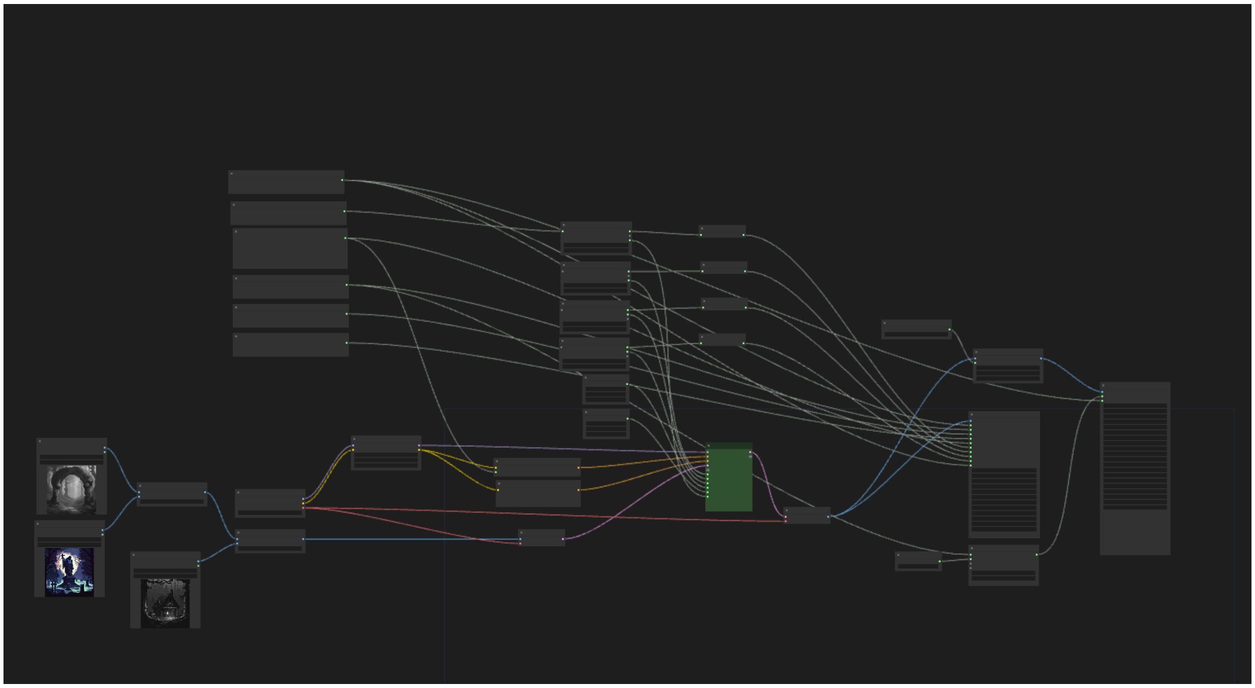


Figure 6: Deployment of UI workflow on AWS Cloud

Workflows

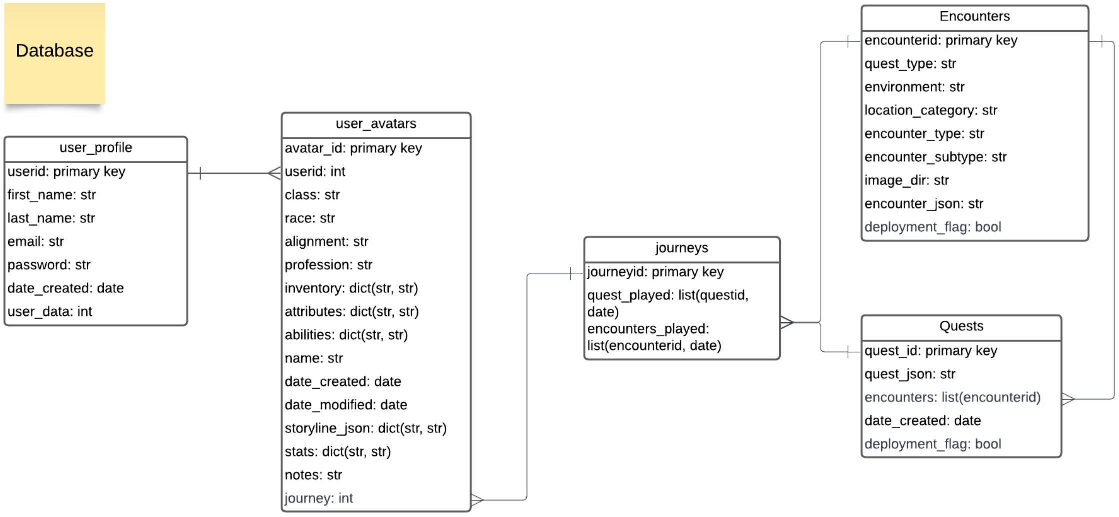
Workflows are generated based off the cloud deployed WebUI tool and saved to be saved and fed to our Image API engine.

Make custom requests to our in-house team for custom workflow nodes.

Settings

- Scheduled time for scheduler to begin image generation and duration
- Hook up workflow saved

Database Design



Database

user_profile

userid: primary key
first_name: str
last_name: str
email: str
password: str
date_created: date
user_data: int

user_avatars

avatar_id: primary key
userid: int
class: str
race: str
alignment: str
profession: str
inventory: dict(str, str)
attributes: dict(str, str)
abilities: dict(str, str)
name: str
date_created: date
date_modified: date
storyline_json: dict(str, str)
stats: dict(str, str)
notes: str
journey: int

journeys

journeyid: primary key
quest_played: list(questid, date)
encounters_played: list(encounterid, date)

Encounters

encounterid: primary key
quest_type: str
environment: str
location_category: str
encounter_type: str
encounter_subtype: str
image_dir: str
encounter_json: str
deployment_flag: bool

Quests

quest_id: primary key
quest_json: str
encounters: list(encounterid)
date_created: date
deployment_flag: bool

API Design

All class diagrams, how they connect and code deep dives go here

Narrative Generation

Image Generation

Logging

Monitoring

Deployment steps

AWS Deployments

Run in terminal from /generator dir
## requirements:
1. install docker
2. set up ssh keys in github repo
3. terminal/putty app
### connect to aws shell:
1. download and move "ManaBurn-EC.pem" from \generators\keys to a directory of your choice
2. in terminal:
- navigate to directory with your "ManaBurn-EC.pem" file
- type "ssh -i "ManaBurn-EC.pem" ubuntu@ec2-52-53-184-199.us-west-1.compute.amazonaws.com"
3. you are now connected to our AWS EC2 shell in AWS
- type "su - moroluckydragon"
- type "docker ps" to show all docker containers running
- the below will show how to deploy each api service
### other docker commands of note:
docker images
docker rmi <image id>
docker ps
docker volume ls
docker volume rm <volume_id>
docker scout quickview

### if you run out of space, clear cache and memory:
docker system prune -a
docker images -a
docker images prune -a
docker container prune
docker volume ls
docker volume prune
docker builder prune
docker system df

## summary of all available api services

api service	build path	recommended container name	recommended build name	recommended port	functionality
ai engine	narrative_service/ai_engine_api/Dockerfile	ai_engine	ai_container	8101	ai chatgpt narrative generator- generates prompts and inserts in AWS DynamoDB
db viewer	narrative_service/database_api/Dockerfile	dbviewer	db_container	5301	db viewer lists entries in AWS DynamoDB

cdn api (deprecated)	narrative_service/encounters_api/Dockerfile	cdn_api	cdn_container	8301	dsyfunc- do not use
-------------	---	---------------------------	------------------------	------------------	--

Template deployment commands:

```
docker build -f <build path> -t <build name> .
docker run -d -p <port>:<port> --name <container name> <build name>
```

service	deployment/stop commands	test urls	logs
ai narrative	docker build-f narrative_service/ai_engine_api/Dockerfile-t ai_engine .	http://localhost:8101/docs	docker logs ai_container
	docker run-d-p 8101:8101--name ai_container ai_engine	http://localhost:8101/generate-narrative/monster	
	docker stop ai_container	http://localhost:8101/generate-narrative/resource_cache	
	docker rm ai_container	http://localhost:8101/generate-narrative/trap	
		http://localhost:8101/generate-narrative/puzzle	
		http://localhost:8301/encounters/resource_cache	
--------------	--	--	-------------------------
db viewer	docker build-f database_service/Dockerfile-t dbviewer .	http://localhost:5301/docs	docker logs db_container
	docker run-d-p 5301:5301--name db_container dbviewer	http://localhost:5301/encounters/	
	docker stop db_container		
	docker rm db_container		
		http://localhost:8301/encounters/resource_cache	
--------------	--	---	

```
-----------------------------------------------------|------------------------- |
| cdn viewer-- deprecated  | docker build-f
narrative_service/encounters_api/Dockerfile-t cdn_viewer .   |
http://localhost:8301/docs                         | docker logs cdn_viewer     |
|                     | docker run-d-p 8301:8301--name cdn_container cdn_viewer
| http://localhost:8301/encounters/puzzle               |                      |
|                     | docker stop cdn_viewer                                          |
http://localhost:8301/encounters/monster              |                      |
|                     | docker rm cdn_viewer                                            |
http://localhost:8301/encounters/trap                 |                      |
|                     |                                                          |
http://localhost:8301/encounters/resource_cache        |                      |
|--------------           |---------------------------------------------------------------------------- |---
-----------------------------------------------------|------------------------- |
```

Summary of API Services and Ports:

API Name:	Port:	Build Path:	Recommended Container Name:	Recommended Build Name:	Notes:
gpt_service	8101	narrative_service/narrative_service/Dockerfile	narrative_container	narrative_api	Narrative text generator that interfaces with LLMs
image_service	7101	narrative_service/narrative_service/Dockerfile	image_container	image_api	Image generator that interfaces with Stable Diffusion
user_profile_service	9101	user_profile_service/narrative_service/Dockerfile	user_container	user_api	Stores all user profile data
encounter_service	7101	encounter_service/narrative_service/Dockerfile	encounter_container	encounter_api	Encounter data for ann instant encounters
quest_service					All quest data
avatar_service					All character avatar data

Testing

Setup

Set up Postman
https://www.postman.com/downloads/

For creatives

For developers

For QA

Appendix

Sample JSON messages:

Encounters: Monsters

```
{
             encounterid: "1dca2811-7405-4af6-ae30-bb7c1d6fcf6d",
             quest_type: "Random_Encounter",
             environment: "Hills_region_highlands",
             location_category: "ancient ruin",
             encounter_type: "monster",
             encounter_subtype: "None",
             image_dir: "https://d1fwhw1a19xrr1.cloudfront.net/1dca2811-
      7405-4af6-ae30-bb7c1d6fcf6d",
             narrative_json:  {
                    location_category: "ancient ruin",
                    environment: "Hills_region_highlands",
                    description: "You've happened upon an ancient tomb
             nestled high in the ominous mist-draped hills. A foreboding
             chill lingers in the air, and the eerie whispers promise
             danger. Suddenly, an Adult Green Dragon materializes from the
             shadows.",
                    success_msg: "Striking with courage, your attack lands
             true, downing the draconic beast and allowing you peace within
             the desolate ruin.",
                    title: "The Whispering Tomb",
                    error: "None",
                    uuid: "1dca2811-7405-4af6-ae30-bb7c1d6fcf6d",
                    encounter_subtype: "None",
                    quest_type: "Random_Encounter",
                    encounter_type: "monster",
                    encounter_desc: "Adult Green Dragon",
                    fail_msg: "The dragon's flame engulfs you as your
             strength fails, mercilessly ending your journey in the
             whispering tomb."
             }
}
```

Encounters: Puzzle

```
{
            encounterid: "70689e5a-1749-4661-ae98-b437ced5956c",
            quest_type: "Random_Encounter",
            environment: "Ocean_island",
            location_category: "ancient ruin",
            encounter_type: "puzzle",
            encounter_subtype: "None",
            image_dir: "https://d1fwhw1a19xrr1.cloudfront.net/70689e5a-
      1749-4661-ae98-b437ced5956c",
            narrative_json:  {
                  location_category: "ancient ruin",
                  environment: "Ocean_island",
                  description: "Upon a weather-beaten island within a vast
            ocean, you discover an ancient ruin. At its heart, a hollow
            statue stands, murmuring riddles with every crashing wave. A
            deep mystery lingers, waiting to be solved.",
                  success_msg: "None",
                  title: "Whispers from the Deep",
                  error: "None",
                  uuid: "70689e5a-1749-4661-ae98-b437ced5956c",
                  encounter_subtype: "None",
                  quest_type: "Random_Encounter",
                  encounter_type: "puzzle",
                  encounter_desc: "Decipher a riddle from murmuring ocean
            waves",
                  fail_msg: "None"
            }
}
```

Encounters: Trap

```
{
            encounterid: "1fa8b3c8-0883-4ef5-ba2d-e074a873f727",
            quest_type: "Random_Encounter",
            environment: "Mountain_peaks",
            location_category: "castle",
            encounter_type: "trap",
            encounter_subtype: "None",
            image_dir: "https://d1fwhw1a19xrr1.cloudfront.net/1fa8b3c8-
      0883-4ef5-ba2d-e074a873f727",
            narrative_json:  {
                  location_category: "castle",
                  environment: "Mountain_peaks",
                  description: "You find yourself in a hidden castle
            nestled among treacherous mountain peaks. Before you, an
            imposing door looms, shrouded in mist, marked with age-old
            draconic glyphs hinting at a perilous trap and an intriguing
            puzzle to unravel.",
                  success_msg: "You decipher the glyphs, disarming the
            trap, and the ancient door creaks open, revealing the path
            forward.",
                  title: "Door of the Dragon Whisperer",
                  error: "None",
                  uuid: "1fa8b3c8-0883-4ef5-ba2d-e074a873f727",
                  encounter_subtype: "None",
                  quest_type: "Random_Encounter",
                  encounter_type: "trap",
                  encounter_desc: "Decode the draconic glyphs",
                  fail_msg: "Misinterpreting a glyph, you trigger the trap,
            a blast of arcane energy strikes you, halting your progress."
            }
}
```

Encounters: Resource

```
{
            encounterid: "5462c903-bc4f-47db-a313-13a069363ced",
            quest_type: "Random_Encounter",
            environment: "Hills_region_vales",
            location_category: "ancient ruin",
            encounter_type: "resource",
            encounter_subtype: "None",
            image_dir: "https://d1fwhw1a19xrr1.cloudfront.net/5462c903-
      bc4f-47db-a313-13a069363ced",
            narrative_json:  {
                  location_category: "ancient ruin",
                  environment: "Hills_region_vales",
                  description: "Within the tranquil folds of a verdant
            vale, you discover ruins; stones whispering tales of a
            forgotten civilization. Amidst the crumbling structures, a
            strange iridescent vein catches your eye.",
                  success_msg: "You skillfully mined 3 units of Emerald!",
                  title: "Veins of the Verdant Vale",
                  error: "None",
                  uuid: "5462c903-bc4f-47db-a313-13a069363ced",
                  encounter_subtype: "None",
                  quest_type: "Random_Encounter",
                  encounter_type: "resource",
                  encounter_desc: "Harvest a hidden resource",
                  fail_msg: "Alas, your pickaxe slips, striking a resonant
            note on the vein and cracking it, an unfortunate turn indeed."
            }
}
```