# AWS GameDay Quest Developer Guide
# Provided Under NDA

Table of Contents







aws



# Introduction

This Developer Guide is intended for technical audiences developing partner-integrated AWS GameDay Quests. For best results, knowledge of AWS CloudFormation and AWS Identity and Access Management (IAM) is recommended. Though a range of programming languages and compute technologies are compatible with the AWS GameDay Quests API, this Developer Guide includes examples and design patterns using serverless technologies including AWS Lambda, Amazon Simple Notification Service (SNS), and Amazon DynamoDB with code samples in Python 3.

# AWS GameDay Overview

## AWS GameDay

GameDay is a collaborative learning exercise that tests skills in implementing AWS solutions to solve real-world problems in a gamified, risk-free environment. This is an interactive opportunity for technical professionals to explore AWS services, architecture patterns, best practices, and group cooperation. GameDay events are intended to be open-ended, with an element of ambiguity. Participants are encouraged to explore documentation and come up with creative solutions to the situations, limitations, and challenges presented to them. GameDay events are competitive, with teams of up to four participants having diverse backgrounds competing with each other to score points and win prizes. For more information about AWS GameDay, visit https://aws.amazon.com/gameday/.

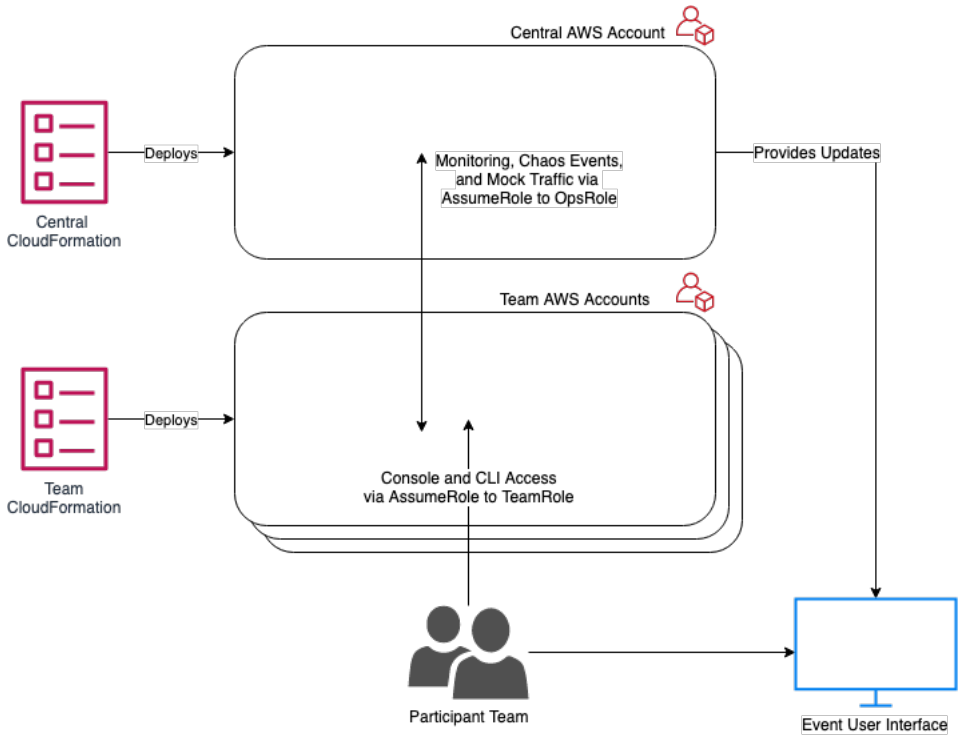
## AWS GameDay Quests

AWS GameDay Quests are independent and modular challenges included in AWS GameDay events. With Partner-developed Quests, Partner-integrated GameDays can feature your AWS solution within the context of the overall GameDay event scenario. This allows participants to get hands-on with your services and directly experience your better together story with AWS. You can tune your AWS GameDay Quest for your customer education objectives and best practices. In addition to generating leads, GameDay provides a memorable education experience for customers.

## Architecture

Each team of participants is provided with an AWS account to share. This account typically contains some initial cloud resources, such as an anti-pattern or incomplete implementation of the desired solution. Participants gain access to their team's account via the AWS GameDay User Interface (UI). This UI also gives teams information about the challenges to solve, allows them to track progress and their score, and provides a means for providing answers to questions.  The event UI can differ depending on which game scenario the event is showcasing.






Text

Participants interact via the AWS Console or [AWS Command Line Interface](#) (CLI) using the *TeamRole* AWS IAM Role. Apart from changes directly made by participants in this account, an AWS GameDay event includes a Central AWS Account. This account keeps monitors the state of participant accounts, keeps track of the overall status of the event, generates mock traffic, provides updates to the event UI, and can make additional changes to teams' AWS Accounts. These changes can include surprise setbacks and challenges to teams (chaos events), as well as assisting in securely facilitating external integrations, such as with a partner service.

The resources in both the Central Account and the Team Accounts are managed via two CloudFormation templates, which you will develop for your Quest. Your Quest is comprised of these two CloudFormation templates and any static assets required by the resources contained in those templates. Examples of these static assets include configuration files, images, HTML, markdown, and deployment bundles for custom applications. In addition to these technical artifacts, your Quest should include documentation on:

- The permissions needed by participants to complete your Quest
- A Guide for AWS GameDay event operators running your Quest
- A Guide for Support staff to help participants with issues or questions about your Quest

# Getting Started

**Note**

The Quest Development Kit ("QDK") is provided by AWS "as is." AWS makes no warranty as to the QDK's use or accuracy, and the QDK is subject to updates without notice at AWS's sole discretion. AWS expressly disclaims all liabilities arising from your use of the QDK, and you remain fully responsible for all content that you create using the QDK including AWS resource costs associated with the QDK and your content.

## Development Environment

This guide assumes that developers have a development environment configured with the following tools:

1. Python 3, including *pip*, and a virtual environment managed with tools like *virtualenv, venv,* or *pipenv*.
2. A *zip* and *unzip* command line utility, such as the Homebrew Formulae[1] for MacOS:

   ```
   $ brew install zip
   $ brew install unzip
   ```

   Or packages for Ubuntu[2] or your preferred Linux distribution on your virtual machine or Windows Subsystem for Linux[3]:

   ```
   $ sudo apt-get install zip unzip
   ```

3. Programmatic access to the AWS Account(s) you plan to use. On a laptop or virtual machine, this can be achieved via the AWS CLI. For more information, see: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html. If you are developing from an EC2 Instance[4] or AWS Cloud9[5] environment, these permissions can be achieved using Instance Profiles.

It is also assumed that in your AWS Account you have access to create, read, and modify the resources necessary to deploy the AWS GameDay Quest Development Kit (QDK) as well as the resources you plan to use in your Quest scenario. Key services to consider:

- Amazon API Gateway
- AWS AppSync

[1] For more information, see: https://brew.sh/
[2] For more information, see: https://ubuntu.com/
[3] For more information, see https://docs.microsoft.com/en-us/windows/wsl/about
[4] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html
[5] https://docs.aws.amazon.com/cloud9/latest/user-guide/credentials.html



- Amazon CloudFront
- Amazon CloudWatch
- AWS CodePipeline
- AWS CodeBuild
- AWS CodeDeploy
- Amazon Cognito
- Amazon DynamoDB
- Amazon EventBridge
- Amazon ECR
- AWS Fargate
- AWS IAM
- AWS Lambda
- Amazon S3
- AWS Secrets Manager
- Amazon Simple Notification Service
- AWS Systems Manager

Some global environment variables to consider for your development environment are:

| | |
|---|---|
| AWS_DEFAULT_REGION | The region where Quest development related resources will be deployed, including the QDK CloudFormation Stack.<br><br>Example:<br>`$ export AWS_DEFAULT_REGION=us-east-1` |
| AWS_GAMEDAY_PROFILE | **[Optional]** The AWS CLI profile to use when making AWS API calls. For more information, see: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html. If not identified, your default profile, or Instance Profile will be used.<br><br>Example:<br>`$ export AWS_GAMEDAY_PROFILE=gameday` |
| QDK_ASSETS_BUCKET | The bucket where you will deploy your Quest data<br><br>Example:<br>`$ export QDK_ASSETS_BUCKET=my-unique-gameday-bucket-name` |

The QDK includes several command line tools that assist with several common tasks, such as deploying/un-deploying the QDK to your AWS account, enabling a Quest, and resetting a Quest. These command line tools are included with the aws_gameday_quests Python package that will be installed in your Central account lambda virtual environment.



**Note**

It is highly recommended to use a virtual environment. If you do not, if your user does not have write privileges to the system's Python site packages location, the installation of the package may fail.

## Prepping Your AWS Development Account

As you go through this guide, if you run into any issues, you can search for the error in the ***Troubleshooting*** section at the end of this document. Before deploying the QDK to your AWS Account, we'll need to create an S3 Bucket to deploy the technical artifacts discussed in ***Architecture*** above. For more information on creating an S3 Bucket via the CLI or AWS Console, see: https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html.

**Note**

Your S3 bucket needs to be deployed in the same region as your QDK environment.

## Reference Quest

Now that your development environment is ready, you can deploy and interact with the reference example Quest provided with this guide. This is an optional step you can skip if already familiar with Quest development.

Save the zip file to your desired workspace directory and unzip it:

```
$ unzip reference_quest.zip
```

Next save a copy of *dev-central-cfn.yaml*, provided along with this guide in the same directory where the unzipped *reference_quest* directory is.

If you list the content of your download folder, you should see something like this:


To access the QDK command line tools, we will need to configure the python virtual environment:

```
$ cd reference_quest/central_lambda_source
$ python3 -m venv .venv
$ . .venv/bin/activate
$ pip install --no-cache-dir -r requirements.txt
$ cd ..
```

Next, export the *QDK_ASSETS_BUCKET* environment variable with the name of the S3 bucket you created earlier. Note you might have set this already a few paragraphs above. This environment variable is used in the *BUILD_QUEST_BUCKET_NAME* variable of the file *package_quest_init.sh*.

```
#!/bin/bash

QUEST_ROOT_DIR=${PWD}
BUILD_QUEST_NAME=reference_quest
BUILD_QUEST_ID=2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63
BUILD_QUEST_BUCKET_NAME=${QDK_ASSETS_BUCKET}
# Include trailing / if a value is defined!
BUILD_QUEST_BUCKET_PREFIX=
```

The *package_quest_init.sh* file uploads a zip file of your Quest's files (i.e. gdQuests-[QUESTID]-quest-artifacts.zip), excluding any dependencies (e.g. binary, .venv, node_modules folders). This zip will be used to seed a CodeCommit repository when the Quest is enabled[6].
The *package_quest_init.sh* file should also upload separately any other static files needed by your Quest such as datasets, binaries, etc, including any large files that cannot be packaged in the CodeCommit zip file.

Using the AWS CLI, we can now upload *dev-quests-data.json* to the Quest's Bucket Prefix configured above:

```
$ aws s3 cp dev-quests-data.json s3://${QDK_ASSETS_BUCKET}/dev-quests-data.json
```

And then package up our Quest and upload it to the same location using our packaging script.

```
$ chmod u+x ./package_quest_init.sh
$ ./package_quest_init.sh
```

## QDK

Assuming you created a virtual environment as instructed in the previous, we are ready to deploy the QDK using the *manage-qdk* utility:

```
(.venv) $ manage-qdk -deploy --template-file ../dev-central-cfn.yaml --s3-target
${QDK_ASSETS_BUCKET}/ --dev-assets-bucket ${QDK_ASSETS_BUCKET}
Deploying QDK
Uploading CloudFormation template to https://my-unique-gameday-bucket-
name.s3.amazonaws.com/example-quickstart/dev-central-cfn.yaml
Upload complete
Creating AWS GameDay QDK stack gameday-qdk-2021-08-29
CreateStack response {"StackId": "..."}. Waiting for completion
Verifying CREATE_COMPLETE
Success.
Generating EventUI credentials.

        ------------------------
        EventUI URL   : https:// abcdefghiklmn.cloudfront.net/
        Table Number : 1
        Password      : PASSWORD
```

[6] Please note that the resulting zip must not exceed 6MB.



**Note**

This deployment may take several minutes to complete. You can follow progress, or identify any issues, in the CloudFormation Console[7].

When this completes, note the URL and credentials displayed. These will be used to interact with the reference example Quest in the Event UI. Before we can do this, the Quest must first be enabled within your Quests environment, using the *enable-quest* utility:

```
(.venv) $ enable-quest --quest-id 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63 --team-id
24eec976eca648eab61514b241032041
Quest 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63 is not yet enabled. Enabling globally before enabling
for team 24eec976eca648eab61514b241032041
Quest not available, status: DEPLOYING, waiting 20 seconds for pipeline
Quest not available, status: DEPLOYING, waiting 20 seconds for pipeline
Quest not available, status: DEPLOYING, waiting 20 seconds for pipeline
b'{"message": "Quest (id: 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63) has been enabled."}'
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: DEPLOYING, expecting: IN_PROGRESS waiting 20s before retry.
b'{"quest-team-enable-stack-id": "..."}'
```

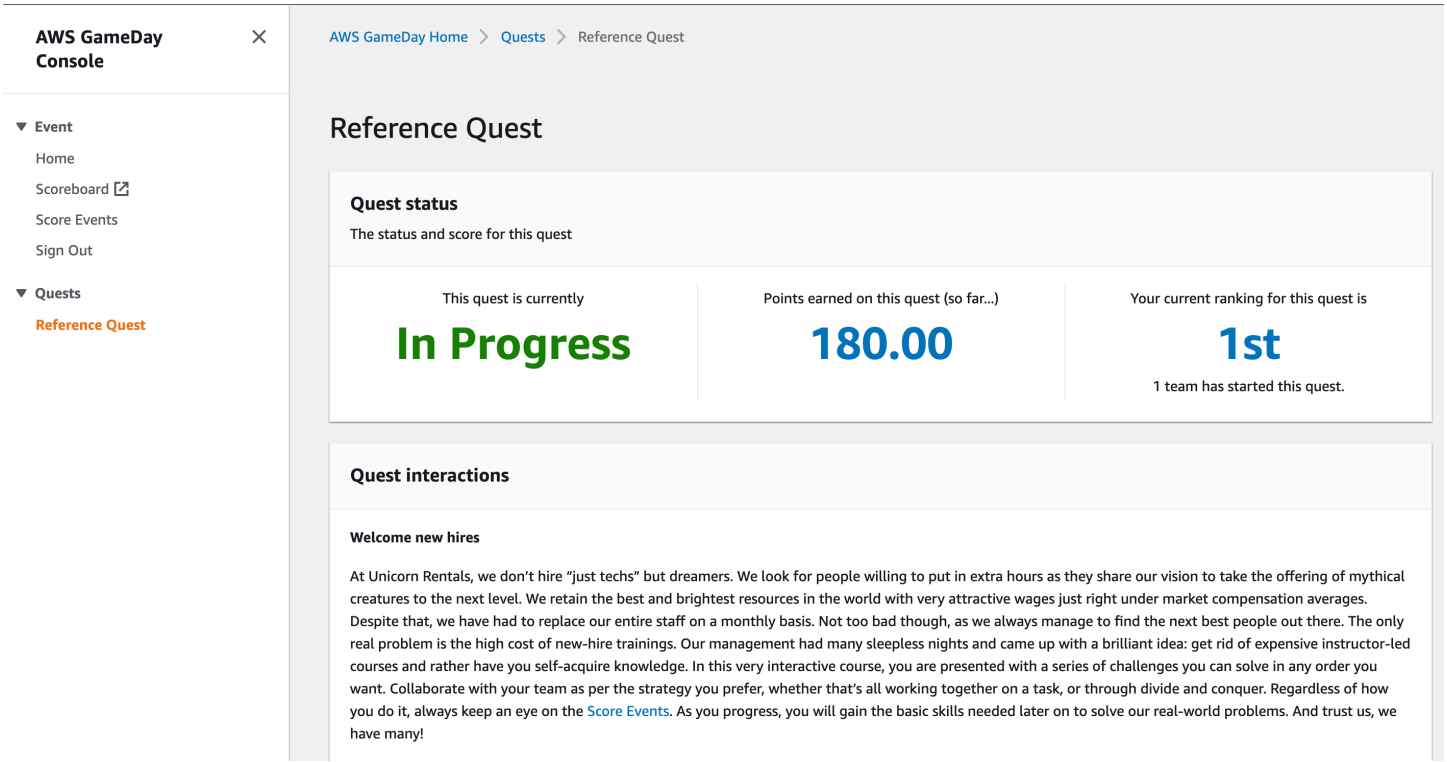
**Note**

Depending on the logging level set, the output of the above command might display several entries for the *get_response* calls.

The *quest-id* used in this command matches the one from the *dev-quests-data.json* file we uploaded to our S3 Bucket, and the *team-id* is the mock Team ID provided with the QDK for testing purposes. This *enable-quest* utility will manage the enablement of Quests by monitoring the progress of the pipelines included in the QDK. Once complete you are ready to launch the Event UI using the link and credentials provided by the *manage-qdk* utility above. Once logged in, click on "*Reference Quest*" in the Quests menu on the left.

[7] https://console.aws.amazon.com/cloudformation/home - Be sure to select your AWS_DEFAULT_REGION






For any of these utilities, you can get more information on the command line options and usage help with the -h argument, for example:

```
(.venv) $ enable-quest -h
```

## Cleanup

If you want to clean up your environment, perhaps for cost saving until you are ready to pick up working with the QDK, you can remove the deployed Quests and the QDK itself from your AWS account.

### Reference Quest Cleanup

Execute the following command the undeploy the reference Quest:

```
$ reset-quest --reset HARD --quest-id 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63
```

**Note**

Replace the Quest id in the above command and execute it if you have other deployed Quests that need removal.

### QDK Cleanup

Execute the following command to undeploy the QDK:

```
$ manage-qdk -undeploy
```



Replace the Quest id in the above command and execute it if you have other Quests deployed that need removal.

# Building Your Quest

The remainder of this guide will walk you through the components of a Quest and how to develop your own from scratch. You can always refer back to the Reference Quest example to see the design patterns discussed in context, or instead of starting from a blank project, make changes to the Reference Quest to experiment with the QDK.

If you had previously removed the QDK from your environment, you will need to install it again as per the Getting Started instructions.

## Setting up Your GameDay Quest Project

As discussed in ***Development Environment*** above, this guide assumes the use of Python 3 with a virtual environment. The directory structure and specific assets included in your Quest are flexible, however we will provide an example structure following the reference implementation example Quest provided along with this guide.

Example Structure:

- *MyQuest/* - Project root directory
    - *artifacts/* - Additional artifacts if needed by your Quest, websites, images, readmes, etc.
    - *build/* - Temporary location for storing deployment artifacts
    - *central_lambda_source/* - Source code for Central Account Lambda Functions
    - *team_lambda_source/* - Source code for Team Account Lambda Functions (if any)
    - *central_cfn.yaml* – CloudFormation Template for Central Account Resources[8]
    - *dev-quests-data.json* – The Quest's specifications in a json format
    - *iam.txt* – Documentation on the permissions needed by participants
    - *package_quest_init.sh* – Shell script to upload your Quest to S3
    - *package_quest.sh* – Shell script to bundle up and deploy your Quest dependencies
    - *team_enable.yaml* – CloudFormation Template for Team Account Resources[9]
    - *readme.md* – High-level information about the Quest

The "central_lambda_source" and "team_lambda_source" directories will have independent virtual environments in our example. In most cases, there will be no need for Team Account Lambda Functions. In the example below, we configure the Central Lambda source directory with a virtual environment including the Quests API.

[8] Both YAML and JSON are supported, see: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-formats.html for more information

[9] A team_activate.yaml CloudFormation Template may also be provided for resources that can be deployed quickly when participants start a Quest.



**Note**

You can execute the *exit* command in case you have already an active virtual environment from the previous section working with the reference Quest.

```
MyQuest$ cd central_lambda_source
central_lambda_source$ python3 -m venv .venv
central_lambda_source$ source .venv/bin/activate
(.venv) central_lambda_source$ pip install --no-cache-dir -r requirements.txt
```

The key packages to install are determined by *Pipfile* or *requirements.txt*. Both are provided in the reference implementation example Quest. These are:

| | |
|---|---|
| boto3 | The AWS SDK for Python3. For more information, see: *https://aws.amazon.com/sdk-for-python/* |
| botocore | The low-level interface used by boto3. For more information, see: https://botocore.amazonaws.com/v1/documentation/api/latest/index.html |
| requests | A library providing HTTP support. For more information, see: https://pypi.org/project/requests/ |
| aws_gameday_quests | The AWS GameDay Quests API. The URL for this package will be provided alongside this guide. |

## Deploying the Quest Development Kit

Now that your S3 Bucket, project directory, and virtual environment have been configured, it is time to deploy the QDK to your AWS Account. The QDK contains the following components:

1. The Quests API and Quests Orchestration which is responsible for the lifecycle of deploying, enabling, and managing CloudFormation stacks simulating the Central and Team Accounts
2. The Developer UI is a static website hosted from an S3 Bucket with a CloudFront Distribution
3. The *aws_gameday_quests* Python package contains a library to assist developers in writing the code to interact with the Quests API
4. DynamoDB Tables store the virtual event state as well as metadata about the Quests that they are working on within this deployment. These tables can be useful for troubleshooting, but your Quest should not directly interact with them.
5. A Mock SNS Topic is also provisioned, which can be used by Quest resources in the Central Account to respond to events that occur in the API or within the Orchestration. For more information, see ***Lifecycle Management*** below.

The infrastructure deployed by the QDK resides in the Central account:

1. QDK components such as ApiGateway and Fargate which manage state for Event and Quests, as well as provide base infrastructure (e.g. ECR image of base functionality)
2. EventUI components such as CloudFront which serve the UI content to the participants

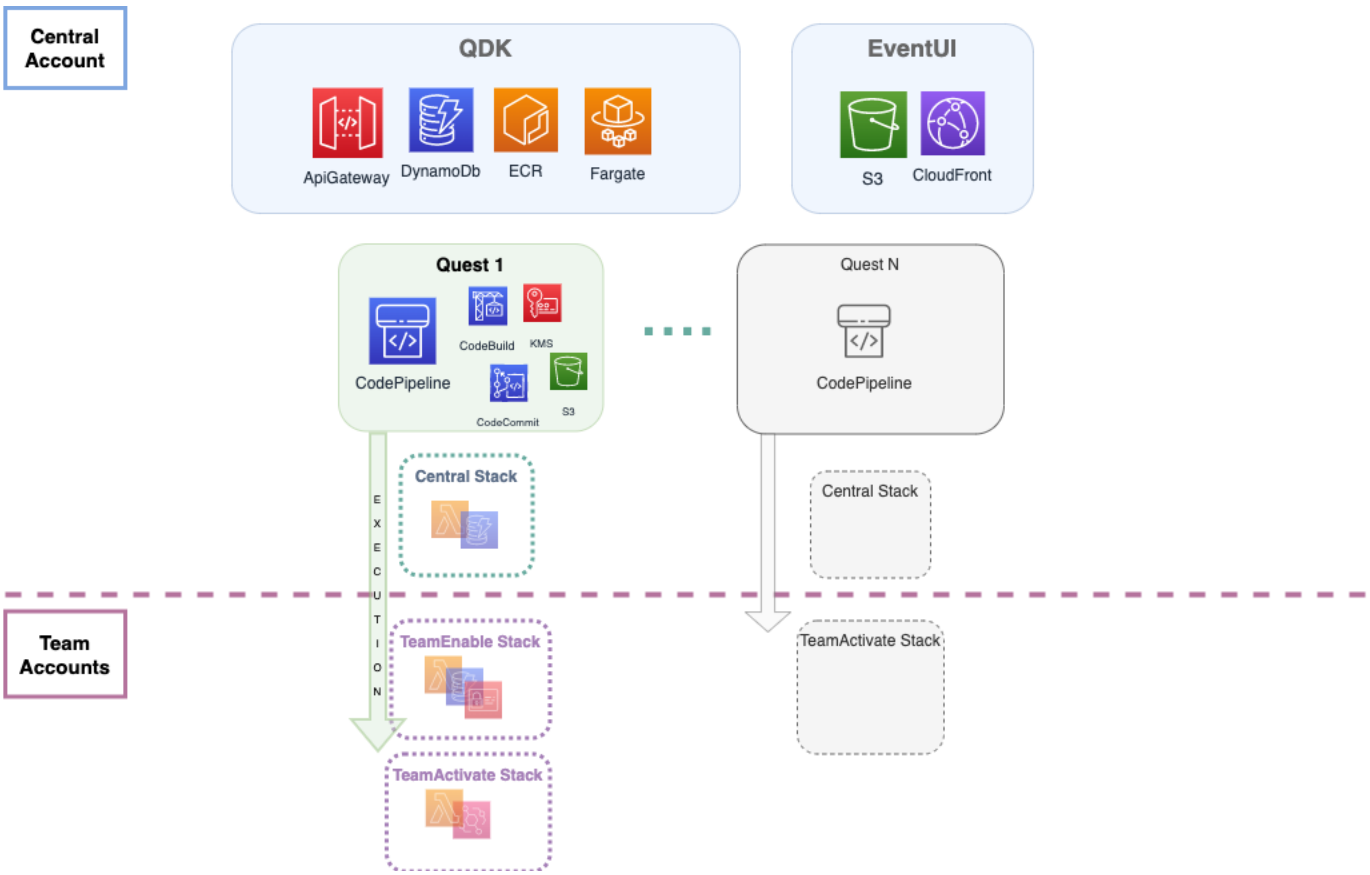
The infrastructure set up for Quests resides in both the Central and Team accounts. Each Quest has two CodePipelines which stand up Quest resources:

1. *QUESTID-quest* CodePipeline stands up the Quest resources
2. *QUESTID-quest-started* CodePipeline initializes a Quest for a team so that they can start playing the Quest



**Note**

The pipelines will be an important service to interact with during development, since they control Quest deployment and hold vital troubleshooting information.




**Note**

In a development environment as described here, the Team Account and Central account are both shared within your AWS Account. There are some limitations in testing edge cases with cross-account AssumeRole operations and possible name space collisions for resources. Keep this in mind while developing your Quest and planning for Integrated Test and Dry Run events.

The *dev-central-cfn.yaml* CloudFormation Stack provided alongside this guide will be used to deploy the QDK. When deploying the stack it is important to be aware of the following Parameters:

| | |
|---|---|
| devAssetsBucket | Identifies the S3 Bucket in your test account where your Quest artifacts will be deployed. If you have |

| | |
|---|---|
| | not yet created this S3 Bucket, please see ***Prepping Your AWS Development Account*** above. |
| devAssetsKeyPrefix | **[Optional]** Identifies the prefix within your S3 Bucket where assets for your Quest will be deployed. |
| devQuestsData | **[Optional]** The name of the JSON file containing metadata about your Quest, which will be loaded by the QDK to populate the mock tables. It is recommended to use the default value, and allow a template *dev-quests-data.json* file to be generated in your S3 Bucket. |
| devDeploymentArtifactSource | Identifies where the Quests API is deployed from, and where metadata and code artifacts are loaded from for the QDK. This value will always be "Development" while building your Quest. |

It is recommended to deploy the QDK via the CLI with a utility provided in the aws_gameday_quests Python package during ongoing Quest development for a streamlined experience and to surface key troubleshooting information and Stack outputs. At this point, you might have already installed the QDK in the **Getting Started** section. However, if not or you undeployed it later on, below is an example command:

```
(.venv) central_lambda_source$ manage-qdk -deploy --s3-target ${QDK_ASSETS_BUCKET}/ --
template-file dev-central-cfn.yaml --dev-assets-bucket ${QDK_ASSETS_BUCKET}
Deploying QDK
Uploading CloudFormation template to https://my-unique-gameday-bucket-
name.s3.amazonaws.com/dev-central-cfn.yaml
Upload complete
Creating AWS GameDay QDK stack gameday-qdk-2021-09-01
Waiting for completion
Verifying CREATE_COMPLETE
Success.
Event UI Dashboard endpoint: https://abcdefghijklmn.cloudfront.net/
Generating EventUI credentials.
b'{"tableNumber": "1", "password": "PASSWORD"}'
```

*The dev-quests-data.json* file is a manifest of Quests available to be deployed and enabled. If you S3 Bucket did not already contain it from the Reference Quest, it should now contain a sample *dev-quests-data.json*. The generated version of this file contains one sample Quest ID that can be used for your Quest.



**Advanced**

If you want to develop multiple Quests simultaneously, you will need to modify this file to add additional Quest stanzas prior to re-deploying the QDK CloudFormation template in your environment. Each Quest should have a unique Quest ID. Alternatively, you can execute the following commands to generate Quest IDs locally:

```
$ python3
>>> import uuid
>>> print(str(uuid.uuid4()))
>>> quit()
```

## Configuring Quest Details

The Quests metadata file, *dev-quests-data.json* generated above contains the following fields:

| Field | Default | Description |
|---|---|---|
| *quest-id* | | Unique identifier (UUID4) for this Quest This is used for Quest API calls. |
| *quest-enabled* | false | Initial enablement status for this Quest in events. Do not change. |
| *quest-central-status* | DISABLED | Initial status for this Quest in DynamoDB state table. Do not change. |
| *quest-name* | | Name for this Quest displayed to participants in the event UI |
| *quest-description* | | The description of this Quest displayed to participants in the event UI |
| *quest-difficulty* | | Difficulty of the event (1-10 scale) to help organizers plan events |
| *quest-use-case* | [] | JSON array of use cases pertaining to this Quest from the AWS “Solutions” menu |
| *quest-services* | [] | JSON array of AWS Services used in this Quest, to help organizers plan events |
| *quest-duration* | 0 | The average time spent by teams completing this Quest, maintained externally |
| *quest-samples* | 0 | The number of times this Quest has been engaged during an event, maintained externally |
| *quest-dependencies* | | Comma separated list of Quest-id’s for other Quests this Quest depends on |



| Field | Default | Description |
|---|---|---|
| *quest-central-cfn* | central_cfn.yaml | The file name for the Central CloudFormation Template as it is stored in the Quest assets S3 Bucket |
| *quest-team-cfn-enable* | team_enable_cfn.yaml | The file name for the Team CloudFormation Template that runs when a Quest is enabled in an event, as it is stored in the Quest assets S3 Bucket |
| *quest-team-cfn-activate* | | The file name for the Team CloudFormation Template that runs when a Quest is started by a team during an event, as it is stored in the Quest assets S3 Bucket. If used, it is recommended to only include resources that deploy in less than five minutes. |
| *quest-readme* | | Reserved for future use |
| *quest-base-points* | 0 | (Deprecated) The maximum number of points awarded for completing this Quest. |

There is no need to modify the default values at this time, but it is important to note the Quest ID created for your Quest. Download your *dev-quests-data.json* via the AWS S3 Console, or with the CLI and look for your Quest ID as below:

```
[
    {
        "quest-id": "acbc1234-567a-bcde-f123-45abc123abcd",
        ...
```

If you had rather generated the Quest ID yourself or did not have the QDK generate the *dev-quests-data.json* file, you can simply add a new entry for your Quest in the DynamoDB *gdQuestsApi-Quests* table. Using the AWS DynamoDB console, *Explore the table items*. Tick the checkbox for one of the entries in the table. Then, *Actions* and *Duplicate item*. Modify the attributes as following:

- Change *quest-id* with the ID of your quest
- Empty the *quest-central-stack-id* and *quest-central-stack-outputs* attributes
- Set *quest-central-status* to *DISABLE*
- Set *quest-enabled* to *False*

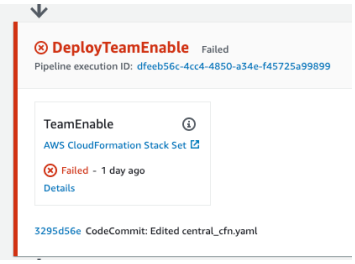
Then, click the *Create item* button.



# Quest Pipelines

As mentioned in the previous section, each Quest has two CodePipeline pipelines.

***QUESTID-quest* CodePipeline**
After executing *package_quest_init.sh* to seed the CodeCommit repository as explained in the ***Getting Started***, you can upload changes to files directly into the CodeCommit repository[10]. This will trigger execution of the pipelines to propagate the changes. ReferWithin the pipeline, you will be able to follow the execution of each action. Should a failure occur, please click on the failed action's Details to troubleshoot:




The pipeline:

1. Is triggered by changes to the CodeCommit repository to execute when a file changes.
2. Deploys the Central stack (CloudFormation template) in the Central account to manage Quest state (e.g. scoring, input/output data in DynamoDb)
3. Deploys the TeamEnable stack set (CloudFormation template), if any, in Team accounts with the resources related to the Quest

***QUESTID-quest-started* CodePipeline**
Once a team has 'started' a Quest, this pipeline will finish initializing the Quest in their team account and enable them to start playing.
The pipeline:

1. Executes when:
    a. there is a file change in the CodeCommit repository
    b. when a team 'starts' a Quest
2. For any team that just 'started' their Quest in the UI:
    a. deploys the TeamActivate stack set (CloudFormation template), if any
    b. updates the status so that the team can get started

You are now ready to begin building the CloudFormation resources and business logic for your Quest.

[10] https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-git-remote-codecommit.html



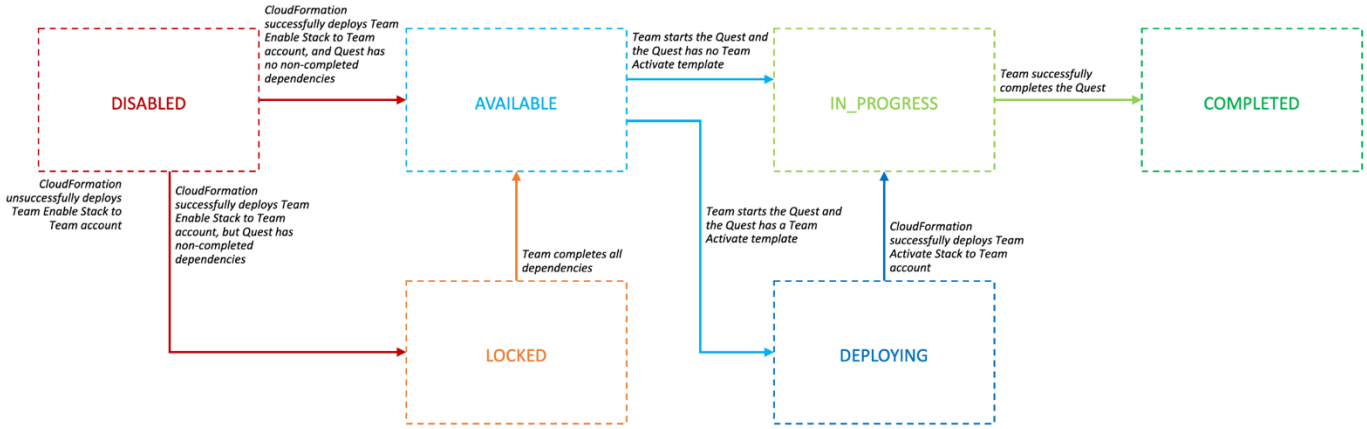
# Central CloudFormation Template

As discussed in ***Architecture***, the Central CloudFormation template contains resources needed to orchestrate your Quest, including tracking participant progress, generating mock traffic, making modifications to participant team accounts, and interacting with the Quests API to connect your Quest to the overall AWS GameDay event containing it. Apart from the examples provided in the reference example Quest, for a full functionality provided by the Quests API, review the documentation in *gdQuestsApi.py* provided with the *aws_gameday_quests* Python package.

## Lifecycle Management

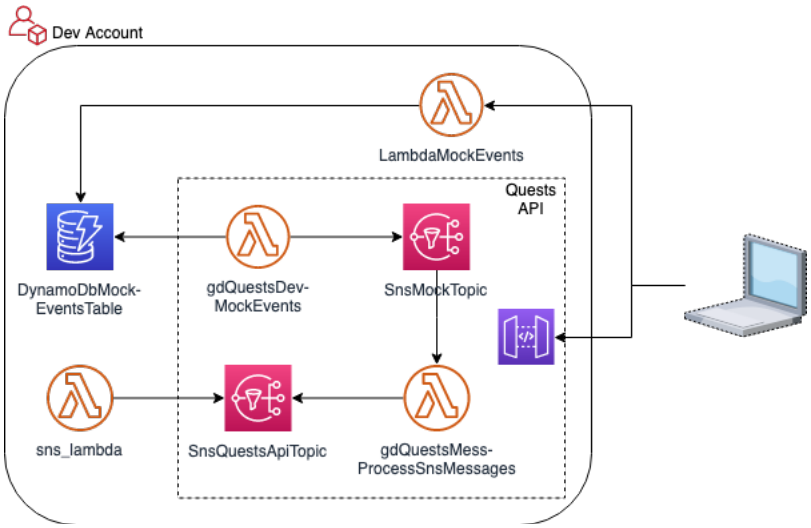
As your Quest is deployed within an AWS GameDay event, and teams interact with it, the Quests API will update its status both globally and for each team that has engaged with it.


The lifecycle of a Quest within the Quests API, beginning with *DISABLED*, is shown above. Once *DEPLOYED*, your Quest is available for teams to enable and begin working on your content. As teams enable and engage with your Quest, it will follow the following lifecycle for each Team:



**Important:** Should there be an issue with deploying a Quest, its CodePipeline will be in 'failed' status in the Central account. Within CodePipeline, you will be able to see which step failed and click into "Details" in order to see its logs and troubleshoot.

The QDK contains an SNS topic used to signal important changes in these lifecycles, as discussed in ***Deploying the Quest Development Kit***. In our reference example Quest, this topic is subscribed to by *SnsLambda* to process changes and orchestrate your Quest. Messages are published to the topic via Quests Orchestration.




**Note**

All Quests share the same SnsMockTopic. It is important to filter messages for your Quest ID by inspecting the message attributes passed to your Lambda Function, stored in `event['Records'][0]['Sns']['Message']`. See *sns_lambda.py* for a detailed example.

## Key status to consider when handling messages from your SnsMockTopic include:

| | |
|---|---|
| QUEST_DEPLOYING | Central and Team "Enable" CloudFormation templates are deploying. Typically used for data or integration related activities not suitable for CloudFormation or custom resources |
| QUEST_IN_PROGRESS | Teams have started the Quest |

| | |
|---|---|
| INPUT_UPDATED | Team has modified an input in the event UI. Typically used to trigger another function to score the team's answers or change the provided URL for a workload the team has created. **Note**: The input value is included in the event under the *value* attribute. |

**Note**

All states are prefixed by *gdQuests:*, for example: *gdQuests:QUEST_IN_PROGRESS*

In the example code snippet below, the SnsLambda triggers an additional Lambda Function to perform Quest initialization actions for a given team, such as adding the team to a DynamoDB table tracking internal progress, or posting a welcome message to the team's event UI.

```
sns_type = event['Records'][0]['Sns']['MessageAttributes']['event']['Value']

if sns_type == 'gdQuests:QUEST_IN_PROGRESS':
    # providing payload for init_lambda
    init_params = {'team_id': team_id}
    lambda_invoke_response = lambda_client.invoke(
        FunctionName=INIT_LAMBDA,
        InvocationType='Event',
        Payload=json.dumps(init_params, default=str)
    )
```

See *central_lambda_source/sns_lambda.py* for more details on this example. Also featured here is a common pattern of using Lambda Function environment variables to pass information about resources created by your Central Template to your business logic. This is defined in the SnsLambda resource of the CloudFormation template:

```
SnsLambda:
  Type: AWS::Lambda::Function
  Properties:
    Handler: sns_lambda.lambda_handler
    Role: !GetAtt LambdaRole.Arn
    Runtime: python3.9
    Timeout: '30'
    Code:
      S3Bucket: !Ref DeployAssetsBucket
      S3Key: !Join
      - ''
      - - !Ref DeployAssetsKeyPrefix
        - !Ref QuestLambdaSourceKey
    Environment:
      Variables:
        QUEST_API_TOKEN: !Ref gdQuestsAPIToken
        QUEST_ID: !Ref QuestId
        QUEST_API_BASE: !Ref gdQuestsAPIBase
        GAMEDAY_REGION: !Ref AWS::Region
        INIT_LAMBDA: !Ref InitLambda
        UPDATE_LAMBDA: !Ref UpdateLambda
        EVENT_RULE_CRON: !Ref EventRuleLambdaCron
```

**Note**



In addition to the Lifecycle state of your Quest, it is often important to ensure that the overall AWS GameDay event your Quest is running within is *IN_PROGRESS* before generating mock load or awarding score. This can be checked via the get_event_status() function in Quests API Client that is a part of the aws_gameday_quests Python package.

While developing your Quest in your AWS Account, it can be helpful to simulate these events. This can be accomplished using the Event UI. See ***Event User Interface*** below.

## Inputs and Outputs

By posting input prompts and markdown-formatted output content, you can dynamically guide participants through your Quest content. Depending on how you'd prefer participant teams to engage, you can provide multiple parallel tasks simultaneously, or one at a time. Output messages can also provide important context to participants such as the IDs of resources deployed via the Team Template (see ***Team CloudFormation Template*** below.)

To post an output to the event UI, use the *post_ouput* function from the Quests API.

```
quests_api_client.post_output(
      team_id=team_id,
      quest_id=QUEST_ID,
      key='call_to_action_1',
      label='Welcome',
      value="Sometimes here at Unicorn.Rentals, it is easier to perform some tasks with
the AWS CLI. Try launching AWS CloudShell from the AWS Console Service menu.",
      dashboard_index=1,
      markdown=True,
)
```

Below is an example of displaying an image on the UI (note the value text starting with '!'):

```
    quests_api_client.post_output(
        team_id=team_id,
        quest_id=QUEST_ID,
        key=output_const.TASK2_KEY,
        label=output_const.TASK2_LABEL,
        value="![My image](http://example.com/my-image.jpg)",
        dashboard_index=output_const.TASK2_INDEX,
        markdown=output_const.TASK2_MARKDOWN,
    )
```

You can control the order in which your inputs and outputs are displayed in the event UI using by setting the *dashboard_index*, with lower values being displayed above larger values. In the event of a tie, an output will appear above an input.

Using inputs, you can collection information from participants or tasking them with finding a value, such as an IP address for an instance. This mechanism also allows participant teams to provide



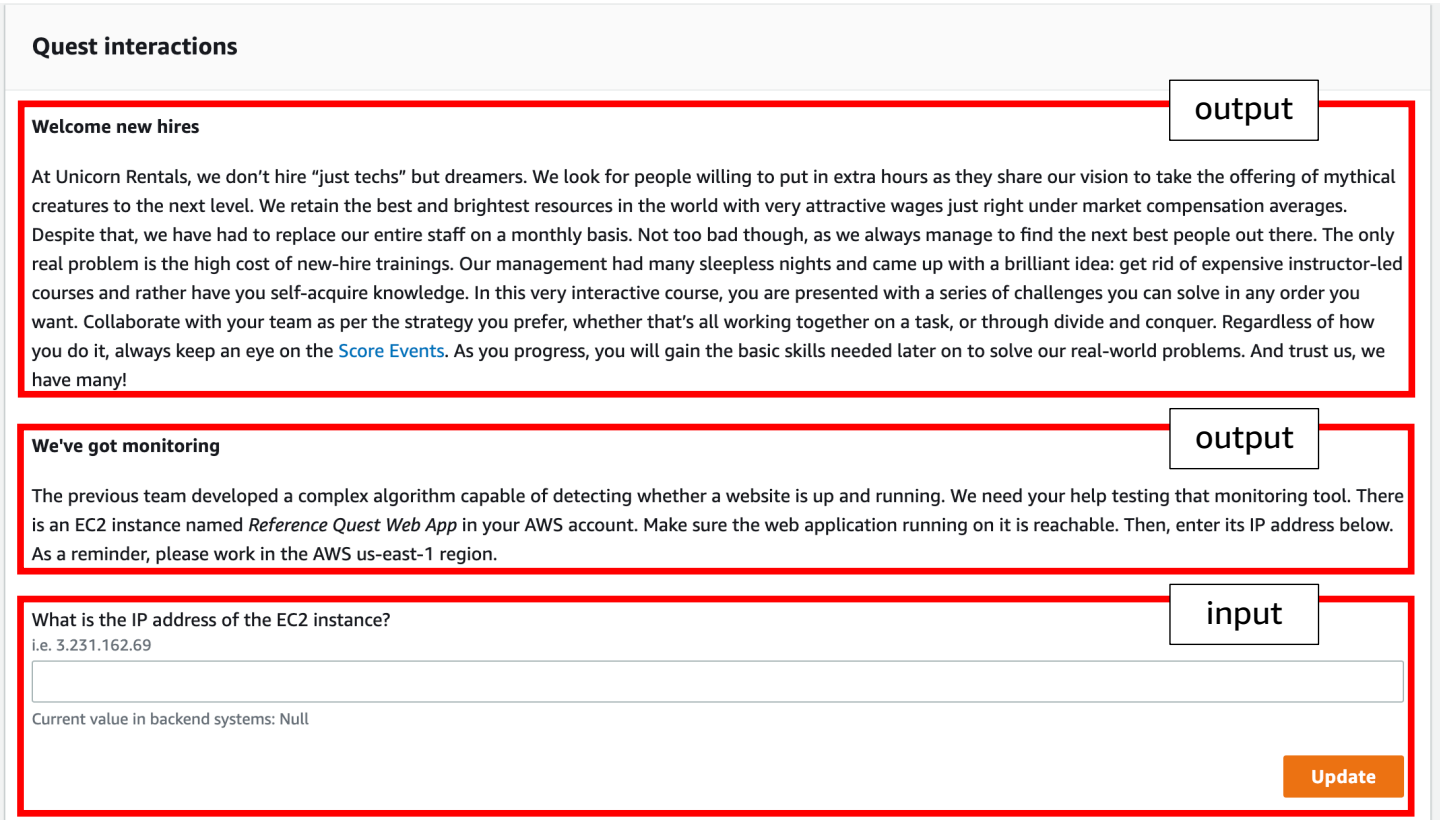
information about resources they have created for your Quest to evaluate, such as the Domain Name of an Application Load Balancer. Once entered, mock traffic can be sent to this address to evaluate the solution.

**Note**

If using this mechanism to collect endpoint information from participants, it is important to verify the data entered independently. Do not trust participant input. Use Boto3 API calls to verify what has been entered is in the Team Account. If this is not done, participants can identify external resources which your Quest could unwittingly disrupt, or which has been created outside of the AWS GameDay event.

To post an input to the event UI, use the *post_input* function from the Quests API. All inputs are treated as strings and should be validated.

```
quests_api_client.post_input(
    team_id=team_data['team-id'],
    quest_id=QUEST_ID,
    key="roleName",
    label="What is the name of the IAM role you're using?"
)
```



**Note**

It is not permitted to collect Personally Identifiable Information (PII), such as email addresses, within your Quest.

To evaluate participant input, we use the same mechanism described in ***Lifecycle Management*** above, and process the answer. In the example below from the reference Quest, we evaluate the participant's answer based on the *gdQuests:INPUT_UPDATED* lifecycle event by invoking another Lambda Function with the related business logic.

```
elif sns_type == 'gdQuests:INPUT_UPDATED':
    key = sns_values['key']
    value = sns_values['value']
    print(f"Quest event: INPUT_UPDATED for team {team_id}, ({key}={value}),
triggering {UPDATE_LAMBDA}...")

    # providing payload for update_lambda
    update_params = {
        'team_id': team_id,
        'key': key,
        'value': value
    }
    lambda_invoke_response = lambda_client.invoke(
        FunctionName=UPDATE_LAMBDA,
        InvocationType='Event',
        Payload=json.dumps(update_params, default=str))
    print(lambda_invoke_response)
```

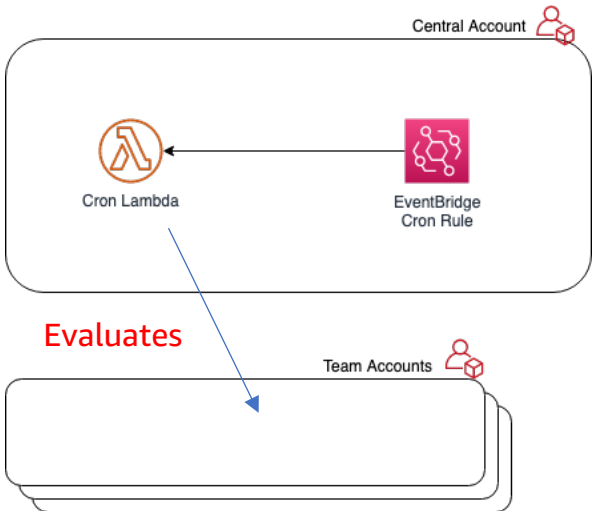
For more details, see *central_lambda_source/sns_lambda.py* and *central_lambda_source/update_lambda.py*.

## Scoring and Evaluation

A significant appeal of AWS GameDay is the open-ended, creative problem-solving participants can engage in when completing your Quest. Scoring is often tied to making a measurement of participant accounts, rather than a direct input. This could include blocking mock malicious traffic, improving the performance of a mock application, or deploying a new resource. To do this, we use cross-account AssumeRole to make AWS API calls to participant accounts, or track state via mock applications deployed in the Central account. In our reference example, this is accomplished via an EventBridge rule running on a schedule, which triggers a Lambda Function.

For more information see: https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-create-rule-schedule.html and *central_lambda_source/cron_lambda.py*.






Using this pattern, business logic triggered by CronLambda can evaluate, make changes to, or send traffic to each participant account and update Quest state information in the Central Account.

**Note**

For large events with more than 20 participant teams, it is important to balance timely responses with resource limitations in the Central Account, such as Lambda concurrency. If your Quest is intended for an event of this scale, consider deploying your logic via an ECS Service, or using Lambda fanout with a multithreading library such as *multiprocessing*[11] to improve concurrency within your Lambda Function.

When your business logic detects that a team has completed a challenge within your Quest and you are ready to award points, you can use the *post_score_event* method of the Quests API to award points. In the example below, 1,500 points are awarded.

```
print(f"CloudTrail lookup result for team {team_data['team-id']}: {cloudtrail_response}")
if len(cloudtrail_response['Events']) > 0:
       team_data['cloudshell-launched'] = True
       quests_api_client.post_score_event(
            team_id=team_data['team-id'],
            quest_id=QUEST_ID,
            description="You successfully launched CloudShell!",
            points=1500
        )
```

When you determine that a team has completed your Quest use the *post_quest_complete* function as in the example below:

```
quests_api_client.post_quest_complete(
            team_id=team_data['team-id'], quest_id=QUEST_ID)
```

[11] For more information, see: https://docs.python.org/3/library/multiprocessing.html



# Team CloudFormation Template

The Team CloudFormation Template is used to provide participants with a consistent initial environment in their Team AWS Account. This allows them to focus on your education objectives rather than provisioning resources or walking through conventional workshop-style prerequisites. Typically the initial state of a Team AWS Account could contain a deployment of a product or service you wish to highlight, or AWS resources following an antipattern architecture for participants to improve, depending on your education objectives.

## Security Considerations

It is important to note that the IAM permissions available to participants in their team accounts are permissive. This is necessary to provide an open-ended experience spanning a wide variety of AWS GameDay content. As a result, all resources and artifacts deployed to a Team account should be considered public and available to participants, including EC2 Instances and Lambda Functions, unless explicitly stated otherwise.

Central accounts should not trust IAM Roles or IAM Users from Team accounts, and the Quests API should not be accessed from these accounts. Additionally, all artifacts deployed in Team accounts should be considered visible to the player, including but not limited to:

- License Keys
- API Keys
- Passwords
- Proprietary applications

Sensitive items of this type should be exclusively employed by the Central account, with actions taken in Team Accounts via cross-account roles. See *assume_team_ops_role()* in the *gdQuestsApi* provided as the *aws_gameday_quests* Python package.

## Integrating Partner Solutions

Using the building blocks discussed in this section, there are a number of ways for partner content to be deployed to Team accounts for a hands-on education experience. Amazon EventBridge integrations[12], external Software as a Service products can be integrated with Team accounts. Solutions can also be deployed to Team accounts using AWS CloudFormation custom resources[13], or via custom business logic deployed to the Central account and initiated by SNS events such as *gdQuests:QUEST_IN_PROGRESS* if secret keys are needed which cannot be visible to participants.

[12] https://aws.amazon.com/eventbridge/integrations/ and https://aws.amazon.com/blogs/compute/using-api-destinations-with-amazon-eventbridge/
[13] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html



Once configured, links to solutions and services including SaaS dashboards and demo account credentials can be shared with participants as outputs.

**Note**

All static assets in your devAssetsBucket ***will be publicly accessible*** during a live AWS GameDay event. To ensure proper handling and deployment of secret keys and passwords, these can be deployed to a protected location like AWS Secrets Manager in the Central account by the event operator. If this mechanism is needed, be sure to document it in your Operator Guide and discuss with the AWS GameDay Team.

### Environmental Considerations

In a live production AWS GameDay environment, the team account has resources provisioned that are not provided by the event runtime and not the QDK. It may be helpful to create these resources outside of your Team Cloudformation Template.

| Resource | Type | Description |
|---|---|---|
| TeamRole | IAM Role | The TeamRole provides the permissions boundaries for participants in your event. The permissions it grants span all Quests and core content in that event. Participants will assume this role when using the STS credentials or console login session provided by the event runtime. |
| ee-default-keypair | EC2 Keypair | This keypair is generated by the event runtime, and provided to players through the event console. It can be referenced in Team CloudFormation templates. This mechanism is supported for compatibility purposes, but the preferred method of providing participants with access to EC2 instances is through EC2 Instance Connect or Session Manager |

# Running Your Quest

To interact with your Quest in your development AWS Account, you will need to package and deploy your Quest assets to your Developer Assets S3 Bucket as created in ***Prepping Your AWS Development Account*** above. The expected structure within your S3 Bucket is as follows:

- *devAssetsKeyPrefix/* - **[Optional]** As defined in your QDK CloudFormation Stack parameters
  - *dev-quests-data.json* – Metadata for your Quest as defined in ***Configuring Quest Details*** above
  - *[Quest ID]/* - Your Quest ID as it appears in dev-quests-data.json
    - *central_cfn.yaml* – Your Central Account CloudFormation Stack
    - *team_enable_cfn.yaml* – Your Team Account CloudFormation Stack
    -

- Other deployment artifacts needed by your Templates, such as Lambda Function zip files.

## Packaging Your Quest

There are no fixed requirements for how you package and upload the other deployment artifacts to your Development S3 Bucket, as long as they are prefixed by the *devAssetsKeyPrefix* you've optionally identified, and your Quest ID as identified above. However, provided in the reference example Quest are two packaging scripts covering many common use cases including Central and Team account Lambda Function code zip files[14].

In *package_quest_init.sh*, configuration variables are defined at the top:

```
#!/bin/bash

QUEST_ROOT_DIR=${PWD}
BUILD_QUEST_NAME=reference_quest
BUILD_QUEST_ID=2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63
BUILD_QUEST_BUCKET_NAME=${QDK_ASSETS_BUCKET}
# Include trailing / if a value is defined!
#BUILD_QUEST_BUCKET_PREFIX=
```

Then, the Quest root directory is zipped up while excluding any built dependencies.

In *package_quest.sh*, source files and dependencies for the Central and Team Lambda Functions are zipped and built separately, this will happen during the Quest's pipeline execution.

**Note**

Verify that the file names created by your packaging script match those referenced by your Central and Team CloudFormation Templates.

## Enabling Your Quest

Before enabling your Quest, ensure that you have deployed the QDK as described in ***Deploying the Quest Development Kit***. If using the CLI option with *manage-qdk*, you will see a CloudFormation Stack in the AWS Console named *gameday-qdk-2022-xx-xx*, matching the date the QDK was deployed. If you deployed your Quest via the AWS Console, it will have the name you provided.

Due to the variety of Quest state transitions for a team, as shown in ***Lifecycle Management*** above, it is recommended to enable Quests via the CLI to manage this orchestration. This can be done via the *enable-quest* utility provided as part of the *aws_gameday_quests* Python package. An example session, enabling the reference example Quest is provided below:

```
(.venv) $ enable-quest --quest-id 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63 --team-id
24eec976eca648eab61514b241032041
Quest 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63 is not yet enabled. Enabling globally before enabling
for team 24eec976eca648eab61514b241032041
Quest not available, status: DEPLOYING, waiting 20 seconds for the pipeline
Quest not available, status: DEPLOYING, waiting 20 seconds for the pipeline
Quest not available, status: DEPLOYING, waiting 20 seconds for the pipeline
b'{"message": "Quest (id: 2ae514a9-a6dc-4fc0-a797-3f4a7bbd1d63) has been enabled."}'
```

[14] See https://docs.aws.amazon.com/lambda/latest/dg/python-package.html for more information



```
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: INITIALIZING, expecting: AVAILABLE waiting 20s before retry.
Quest not ready for team. Status: DEPLOYING, expecting: IN_PROGRESS waiting 20s before retry.
b'{"quest-team-enable-stack-id":...}
```

The Team ID in the above example, *24eec976eca648eab61514b241032041,* is the ID for the mock Team provided by the QDK. To see more metadata about this team, see the *gdQuestsApi-MockTeams* DynamoDB Table in the AWS Console. Once the Quest is *AVAILABLE* or *IN_PROGRESS*, it is ready to be interacted with using the Event UI.

### Fixing Issues with Your Quest

It is likely that as you are testing your Quest locally, you need to fix code issues with it. Then, you will need to redeploy the latest changes and resume testing. The majority of issues can be easily redeployed by re-executing the Quest pipeline. If the change is to a single file, you can just use the AWS CodeCommit console in your account, identify the repository, edit the file, and commit. This will trigger the execution of the pipeline, which will deploy the latest changes.

However, if multiple files were changed, it is recommended to use the *git* tool. Clone the CodeCommit repository to your local machine, for instance using the following command:

```
$ git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/<my-repo>
```

Then, sync the changes in the cloned repo in whichever way you prefer. Then, push the changes, for instance with the following commands:

```
$ cd <my-repo>
$ git add --all
$ git commit -am "fixed a few issues"
$ git push
```

Monitor the AWS CodePipeline as it will run automatically and deploy the latest changes.

If you happened to change files that are only executed when the Quest is started, the init_lambda.py for instance, you will need to reset the status of your Quest in order to test that execution again. Refer to the following **Resetting Your Environment** section for further instructions.

# Resetting Your Environment

In the course of testing your Quest, you will occasionally need to reset the state of your QDK, Quest, or both. To completely redeploy your QDK CloudFormation Stack, including your Quest, use the *manage-qdk* utility provided in the *aws_gameday_quests* Python package. The *-undeploy* command line option will delete AWS GameDay stacks from your account, and if the *-deploy* command line option is provided, redeploy the QDK. Example below:



```
$ manage-qdk -undeploy -deploy --s3-target ${QDK_ASSETS_BUCKET}/ --template-file dev-central-cfn.yaml --dev-assets-bucket ${QDK_ASSETS_BUCKET}
```

From here, you can re-enable your Quest as described in ***Enabling Your Quest.***

In some situations, it may be more expedient to reset the state of your Quest, without redeploying the QDK. This can be done using the *reset-quest* utility provided in the *aws_gameday_quests* Python package. To reset your Quest, within the QDK, to its initial state use the command:

```
(.venv) $ reset-quest --reset INIT --quest-id [your Quest ID]
```

**Note**

This utility is unaware of any state management specific to your Quest, such as additional DynamoDB tables created in your Central CloudFormation Template. You will need to reset these through a mechanism of your choosing.

You can also undeploy your Quest and reset its state within the QDK with the following command:

```
(.venv) $ reset-quest --reset HARD --quest-id [your Quest ID]
```

This method deletes the Central and Team CloudFormation Stacks for your Quest. If your Quest's state management mechanism, such as a DynamoDB Table, is a resource in your Central CloudFormation Stack, your custom Quest state will also be reset.

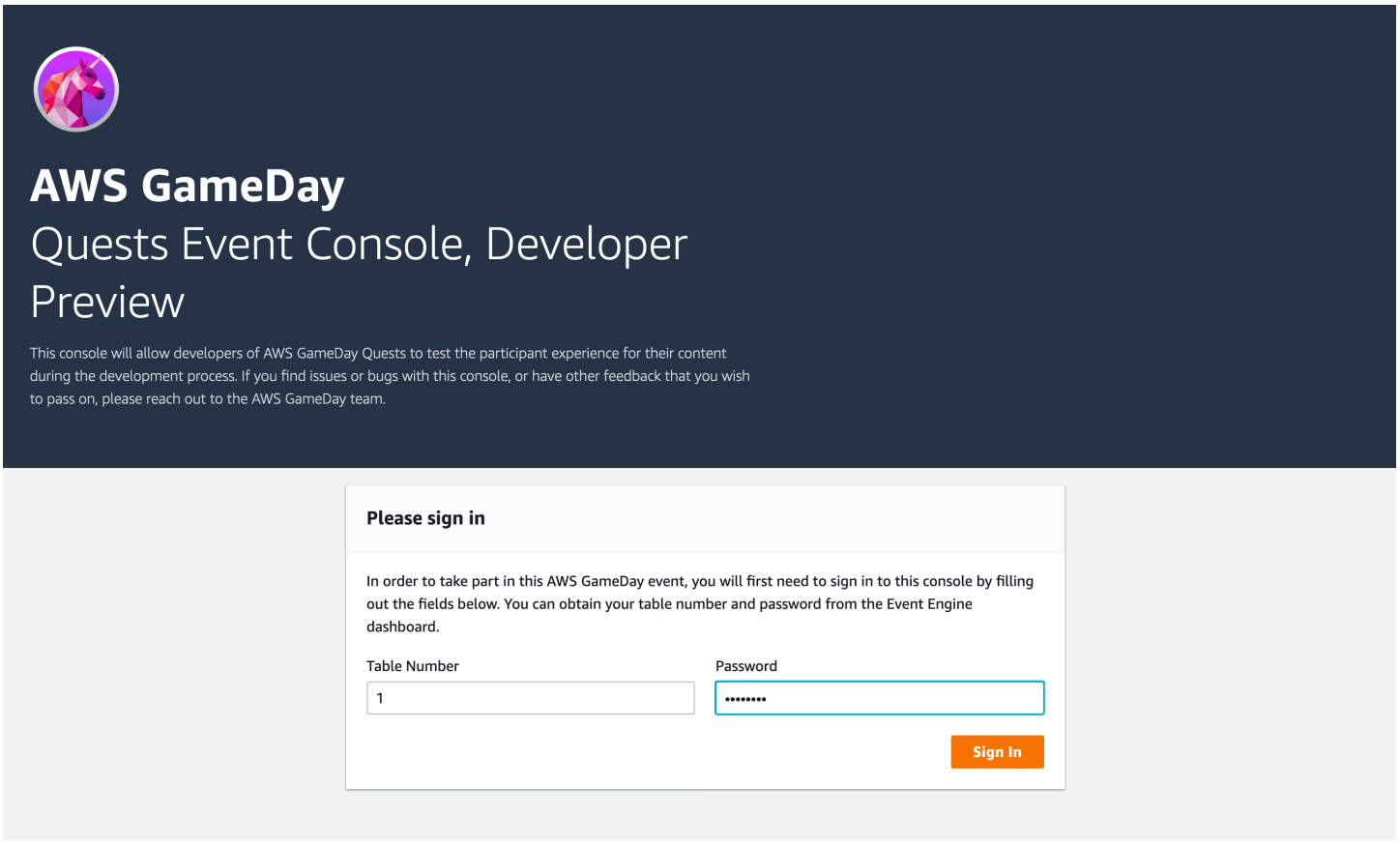
# Event User Interface

The Event UI used for AWS GameDay events featuring exclusively Quest-based content is bundled with the QDK, and deployed to your AWS Account. When deploying the QDK with the CLI with the *manage-qdk* utility function provided with the *aws_gameday_quests* Python package, the CloudFront Distribution URL will be provided along with the credentials needed to access the player dashboard as the Mock team described above in ***Enabling Your Quest***. Example output:

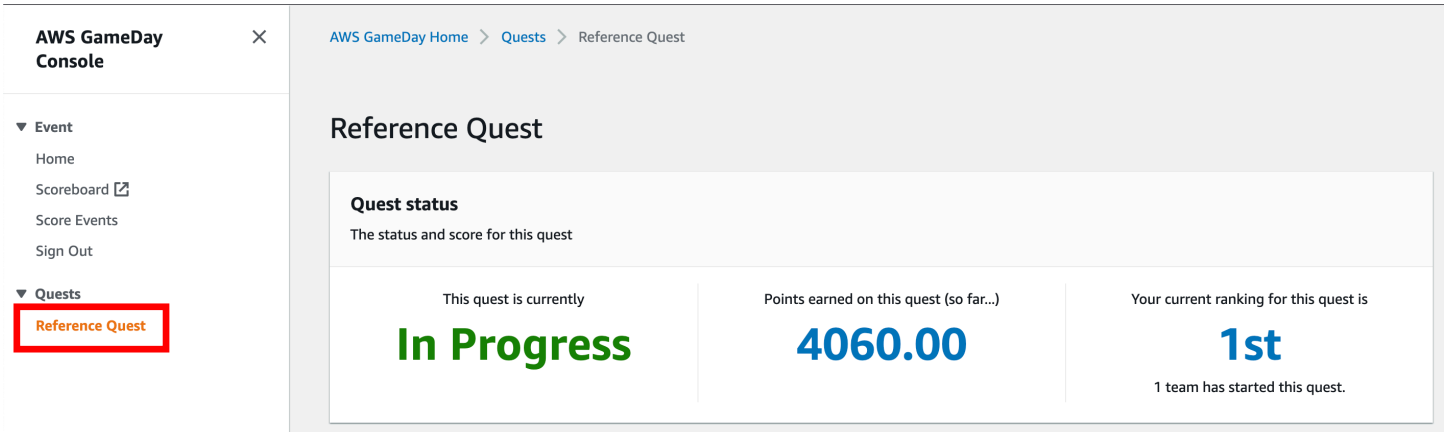
```
Event UI Dashboard endpoint: https://abcdefghijklmn.cloudfront.net/
Generating EventUI credentials.
b'{"tableNumber": "1", "password": "PASSWORD"}'
```

While the QDK CloudFormation Stack is deployed in your AWS Account, you can access this URL via your web browser, and login with these credentials as shown below:






To access your Quest, click on it in the left navigation menu as shown below:




For the best participant experience it is highly recommended to utilize the SNS topic to receive notifications when participants update input interactions. By invoking your Lambda function (and consequently executing your Quest's logic) in response to input interaction updates, any updates to input, output, or hint interactions are reflected in near-real time in the participant's Event UI, resulting in a responsive participant experience.

# Hints

Optionally, your Quest can provide participants with hints to complete complicated challenges. These static hints are defined in your Quest's code. When offering a hint to a team, use the *post_hint()* function in the *GameDayQuestsApiClient* class from the *aws_gameday_quests* python package. The fields that can be passed in are as follows:

| | |
|---|---|
| team_id | The unique string identifying the team |
| quest_id | The unique string identifying the Quest |
| hint_key | A unique string identifying this hint. |
| label | The label for this hint when offered to participants |
| description | A description of what information this hint will provide participants, if accepted and displayed |
| value | The hint text displayed to participants who accept the hint. |
| dashboard_index | The index on the event UI where this hint will be offered and displayed. This allows you to intermingle inputs, outputs, and hints |
| cost | The cost of accepting this hint in points. |
| status | Can be two options:<br>OFFERED: offers the hint to the participant<br>DISPLAYED: displays the hint to the participant |

For more information, see the function documentation for this package.

# Guidelines and Best Practices

Carefully review and implement the following guidelines and best practices into your Quest. This will help provide a consistent experience for participants as they play through different Quests.

## Gender Neutral / Inclusive Language

As a Quest developer, you are responsible for using appropriate wording for your Quest narrative. Examples include but are not limited to:

- Avoid referring to a specific group of people or a nation.
- Characters in your narrative including a CEO or a hacker are not automatically men. Use gender neutral pronouns.
- Refrain from using non-intuitive idioms or slang whose meaning may not be clear to non-native English speakers.

Refer to your organization's *Diversity Equity & Inclusion* guidelines for a better understanding of the matter.



## User Experience / User Interface

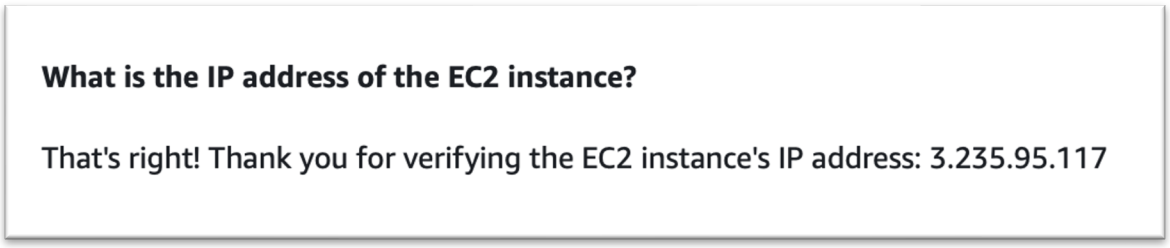
- Add outputs to the Quest UI from top to bottom. Status changes or feedback go at the bottom of the task/challenge they pertain to.
- Use sentence case capitalization for output labels.
- Clearly indicate to teams whenever a task is done, either through proper wording or by adding "(Done)" to the output label. For example:

  **Find our website (Done)**
  ...
  lorem ipsum dolor
  ...
  Well done finding our website.

- Instruct teams to wait while checks are happening in the background and the UI is updating with next step or feedback. For example, the Quest asks the team to launch AWS CloudShell. As logs might take some time to reflect in CloudTrail for validation, tell the team something along the line of "*Once you launch CloudShell, wait while we perform backend verifications.*"
- As teams progress through Quest tasks, do not delete the previously completed tasks from the UI. By looking back through the UI, any member of the team should be able to understand what had been done until that point.
- Input elements require special handling. For instance, if a team has provided a correct answer or an input that must not require a further update, then subsequently delete the input and replace with an output that "simulates" what the Quest's question and the team's answer were. This is to keep a history as per the aforementioned point above. Here is a code example followed by the result in the UI:

```
# Delete input since cannot be updated as task can be started only once
quests_api_client.delete_input(
    team_id=team_data["team-id"],
    quest_id=QUEST_ID,
    key=input_const.TASK1_ENDPOINT_KEY
)
# Replace input with an output to leave a trace of what has been done
quests_api_client.post_output(
    team_id=team_data['team-id'],
    quest_id=QUEST_ID,
    key=output_const.TASK1_IP_ADDRESS_CORRECT_KEY,
    label=output_const.TASK1_IP_ADDRESS_CORRECT_LABEL,
    value=output_const.TASK1_IP_ADDRESS_CORRECT_VALUE.format(team_data['ip-address']),
    dashboard_index=output_const.TASK1_IP_ADDRESS_CORRECT_INDEX,
    markdown=output_const.TASK1_IP_ADDRESS_CORRECT_MARKDOWN,
)
```






- Use an image to convey a complex concept or a funny joke (be inclusive and not offensive), for instance an architectural diagram to overview the current infrastructure setup or the desired state. Do not abuse images. Keep it to a maximum of 2. Also, make sure images are licensed for public domain use by Amazon, or be creative and create your own image.

### Team Enable Template (*team_enable_cfn.yaml*)

These are AWS resources that will be deployed in the team accounts. It is not mandatory but only needed to provision resources specific for the individual teams. If a resource can be shared by all teams without being altered, then consider adding it to the Central template (*central_cfn.yaml*). For example, a source RDS database all teams will read from with read_only permissions.

### Team Assets

These are assets that are needed somewhere in the team accounts. They might be a zip file being referenced by the team template or a resource link on the Quest UI, like in the following example from the reference Quest's *package_quest_init.sh*:

```
echo -e "\nUploading additional Quest artifacts to S3"
cd ${QUEST_ROOT_DIR}
aws s3 cp artifacts/images/curl.jpeg
s3://${BUILD_QUEST_BUCKET_NAME}/${BUILD_QUEST_BUCKET_PREFIX}${BUILD_QUEST_ID}/curl.jpeg ${PROFILE_ARGUMENT}
```

### Init Lambda

This function executes for a team whenever they start the Quest. Therefore, it is executed only once per Quest per team. This is where all the initial Quest instructions are being posted to the UI, team template output values are retrieved (Figure 1), DynamoDB Quest tables are initialized (Figure 2), and custom resources are provisioned if needed.

```
# Retrieve CloudFormation stack outputs
ip_address = cfn_utils.retrieve_team_template_output_value(quests_api_client, QUEST_ID, team_data, "EC2IPAddress")
security_group = cfn_utils.retrieve_team_template_output_value(quests_api_client, QUEST_ID, team_data, "SecurityGroup")
accesskey_value = cfn_utils.retrieve_team_template_output_value(quests_api_client, QUEST_ID, team_data, "UserAccessKeyName")=
```

*Figure 1 - Team template output values lookup*



```
    # Populate the QUEST_TEAM_STATUS_TABLE for this team
    dynamo_put_response = quest_team_status_table.put_item(
        Item={
            'team-id': str(team_id),
            'quest-start-time': int(datetime.datetime.now().timestamp()),
            'ip-address': ip_address,
            'security-group': security_group,
            'is-ip-address': False,
            'is-monitoring-chaos-started': False,
            'is-monitoring-chaos-done': False,
            'is-cloudshell-launched': False,
            'accesskey-value': accesskey_value,
            'is-accesskey-rotated': False,
            'credentials-task-started': False,
            'is-final-task-enabled': False,
            'is-answer-to-life-correct': False,
            'version': 0 # This is for optimistic locking
        }
    )
```

*Figure 2 - DynamoDB table initialization*

## Check Team Lambda

This function executes by default at 1-minute intervals, triggered by Amazon EventBridge. This is where team account validations that require background checks are done. One example is awaiting that a team brings a web application up. The Check Team Lambda performs the check every minute and once successful, it awards or detract points (Figure 3), unlock new tasks, and provide visual feedback on the UI.

```
        print(f"The web application for team {team_data['team-id']} is DOWN")
        quests_api_client.post_score_event(
            team_id=team_data["team-id"],
            quest_id=QUEST_ID,
            description=scoring_const.MONITORING_WEB_APP_DOWN_DESC,
            points=scoring_const.MONITORING_WEB_APP_DOWN_POINTS
        )
```

*Figure 3 - API call detracting points for a team*

## Update Lambda

This function is triggered when a participant enters a value in an input via the UI. In turn, the function will post feedback on the UI, award or detract points, unlock new tasks.

Be aware that a team might click the button multiple times in a just a few seconds apart, thus triggering the function more than once. If the function handles scoring, make sure your logic only executes once. Line 167 below is an example on how to avoid such scenario using a boolean flag and make a DynamoDB update as first thing before continuing processing.



```
    # Task 4 evaluation
    elif event['key'] == input_const.TASK4_KEY:

        # Check team's input value
        value = event['value'].strip().lower()
        if value in input_const.TASK4_KEY_CORRECT_ANSWERS:

            # Correct answer - switch flag to true
            team_data['is-answer-to-life-correct'] = True

            try:
                # First update DynamoDB to avoid race conditions, then do the rest on success
                dynamodb_utils.save_team_data(team_data, quest_team_status_table)
```

*Figure 4 - Handling of team input*

## Team Template Outputs Lookup

When you need to validate team account resources from the central account, a common pattern is to use outputs from the Team Enable template. Figure 5 is an example of CloudFormation outputs in a team template, whereas Figure 1 shows how those outputs are retrieved. The best practice is to perform this retrieval operation in the *Init Lambda*, store the value in DynamoDB, and use it later on in the *Check Team Lambda*, *Update Lambda*, or anywhere else you need.

```
Outputs:
  EC2IPAddress:
    Description: The IP address of the EC2 instance running the web application
    Value: !GetAtt WebAppOnEC2.PublicIp

  SecurityGroup:
    Description: The Security Group ID
    Value: !Ref PublicSecurityGroup

  UserAccessKeyName:
    Description: Access key for the user
    Value:
      Ref: AccessKeys
```

*Figure 5 - CloudFormation outputs*

## dashboard_index

For this parameter of the *post_output* and *post_input* calls, use two-digit numbers, so that it's easier later on to shift things around. Perhaps, you can use the same first digit for the same task. For example, 10 to 19 for task #1's outputs and inputs; 20 to 29 for task #2, and so forth.

## Endpoint Validation

Validating a team's endpoint such as a load balancer or an IP address is usually done in *Check Team Lambda*. This use case is usually a good candidate for chaos events, as to simulate an application



going down for a reason such as a malicious attack or a bad deployment. Figure 6 is an example implementation. However, other libraries or validation techniques can be used.

```
# Checks whether the monitoring web app is up or done and returns True or False respectively
def check_webapp(team_data):
    try:
        print(f"Testing web app status")
        conn = http.client.HTTPConnection(team_data['ip-address'], timeout=5)
        conn.request("GET", "/index.html")
        res = conn.getresponse()
        data = res.read().decode("utf-8")
        print(res.status)
        if res.status != 200:
            raise Exception(f"Web app down: {res.status} - {res.reason}")
        return True
    except Exception as e:
        print(f"Web app not available: {e}")
        return False
```

*Figure 6 - Checking an endpoint status*

## Chaos Events

One aspect unique to the experience of AWS GameDay is the introduction of disrupting events while the team is making progress. Chaos events simulate a real-life situation where, for instance, something suddenly breaks in production or requirements change, and pressure is on to fix it in a timely manner. Typically, points are detracted at 1-minute intervals until the team can successfully mitigate the cause of the chaos event. Refer to the Scoring section below for important considerations to keep in mind while implementing this use case. Figure 7 is an example implementation of checking for the time a chaos event is to be triggered.

```
    # Check if chaos has not started yet and the team has provided the EC2 IP address
    if not team_data['is-monitoring-chaos-started'] and team_data['is-ip-address']:

        # Start chaos event if timer is up or task 2 and 3 are done
        if ((team_data['is-cloudshell-launched']    # Task 2
            and team_data['is-accesskey-rotated'])  # Task 3
            or is_chaos_timer_up(team_data['monitoring-chaos-timer'],int(CHAOS_TIMER_MINUTES))
            ):

            print(f"Time for chaos event for team {team_data['team-id']}")

            # Switch flag that chaos event has started
            team_data['is-monitoring-chaos-started'] = True
```

*Figure 7 - Chaos event example implementation*



## Assume Team Account's Role for Cross-Account Validations and Lookups

The Quest's central account has the ability to assume the *OpsRole* role in the team's account in order to validate an AWS resource or do some lookup. For instance, the Quest asks the team to rotate a user's credentials. Then, the central account's validation will assume the role in the team's account and check the IAM credentials for the user. Figure 8 demonstrates the QDK API call to assume the role in the team's account, whereas Figure 9 is an example of how to use the cross-account session for validation of a team's AWS.

```
# Establish cross-account session
print(f"Assuming Ops role for team {team_data['team-id']}")
xa_session = quests_api_client.assume_team_ops_role(team_data['team-id'])
```

*Figure 8 - Assume role example*

```
# Lookup CloudShell events in CloudTrail
cloudtrail_client = xa_session.client('cloudtrail')
quest_start = datetime.fromtimestamp(team_data['quest-start-time'])
cloudtrail_response = cloudtrail_client.lookup_events(
    LookupAttributes=[
        {
            'AttributeKey': 'EventName',
            'AttributeValue': 'CreateSession'
        }
    ],
    StartTime=quest_start
)
print(f"CloudTrail lookup result for team {team_data['team-id']}: {cloudtrail_response}")
```

*Figure 9 - Cross-account validation example*

## Non-linear Play-Through

This design pattern is preferred over the more linear flow where tasks or challenges are presented to teams in a sequential manner. A non-linear flow might present multiple challenges at once, or any other approach that might not give participants a feeling of participating in a workshop through step-by-step guidance. Moreover, a non-linear approach with multiple tasks that can be worked in parallel allows engaging multiple team members at once on the same Quest.

## Default Resources CloudFormation Lookup

A GameDay event is comprised of multiple Quests, each provisioning its own set of AWS resources. Therefore, developers should be conscious of the AWS quotas in both central and team accounts. For instance, some of the quotas that are known to be more problematic are the maximum number of VPCs, IGWs, and EIPs in an account. Try to reuse default resources as much as possible, that is, the resources that come bundled with a brand-new account (default VPC, subnets, Internet Gateway, etc.). Use a CloudFormation custom resource to lookup default resources in the account. The reference Quest provided with this guide contains resource lookup code that can be or extended as needed.



## Hints

- Present the hint within the task it belongs to, unless it is a generic cross-task hint. In that case, add it at the bottom of the initial output/instructions for the team.
- Do not provide an answer or solution in the hint, but only pointers that help the team progress.
- Delete a Hint only if it was not revealed by the team and it is no longer applicable (i.e., the team has completed that task).

## Scoring

- Always award Quest completion points.
- Quests should implement a mix of continuous scoring and checkpoint scoring, as it delivers a more entertaining GameDay experience and ensures a wide spread of actual team scores with a clear winner.

**Checkpoint Scoring** refers to the Central account Lambda functions awarding the team points just once for a specific action, for instance, the team provided a correct answer, completed a task, or completed the entire Quest. The only use of checkpoint scoring should be avoided, or at very least there should be a mechanism that assigns bonus points based on the time it takes for a team to complete the Quest: the quicker, the more points. This is to avoid ties at the end of the game, but also not to make the play-through experience workshop-like.

**Continuous Scoring** refers to the Central account Lambda functions awarding or detracting the team points on 1-minute intervals (usually done via the *Check Team Lambda* function), for instance as the team successfully brings up an endpoint or while the team has not yet stopped an attack. The number of points given at any interval is usually much lower than the ones from checkpoint scoring, since it's continuous and the total for the team can grow quickly. Do some proper tuning of the scoring.

Continuous scoring requires some thinking and careful implementation to avoid undesired outcomes that might lead to cheating or unfairness. Some scoring patterns you can reuse are described below:

**[No Scoring Until Action is Taken]** No points are accrued when the Quest starts. For points to start either increasing or decreasing, it takes the team to perform some action.

**[Positive Scoring Until Countdown]** At some point while playing a Quest (no matter when), the team starts gaining points at 1-minute intervals because, for instance, "they fixed something". At that time, a countdown starts (say, 5 minutes). Once the countdown ends, either the positive scoring stops or negative scoring starts (i.e., chaos event).

**[Positive Scoring with No Capping]** If a team is gaining points every minute and they delay or do not complete the Quest, by the time the event ends, they might have accumulated more points than other teams who completed the Quest in a timelier manner. To counter-balance



this situation, 1) reward Quest completion points in an amount greater than the sum of all interval points; 2) reward Quest completion **bonus** points as a function of the time it takes for teams to complete the Quest: the faster, the more points.

**[Negative Scoring at Start]** Points are detracted at regular intervals as soon as the team starts working on a task (i.e. there is an ongoing attack that needs immediate mitigation). It is recommended that an initial sum of points be granted as the Quest starts to avoid that the team goes minus-zero points immediately and experience anxiety or frustration. The number of points detracted at every interval should be so that the team would not go negative even if they were not to take any action until the event ends. Use proper wording to catch the team's attention that something needs immediate action in order to avoid scenarios where a Quest is started and then ignored in favor of another one. Another approach is to have the team read some initial instructions without giving too much away. Then, ask them whether they are ready to take on the challenge and warn points will be lost until the problem is solved. This is the approach implemented in the Task #3 of the Reference Quest. Figure 10 is the moment before the team takes action whereas Figure 11 is after the team decided to start working on the task.

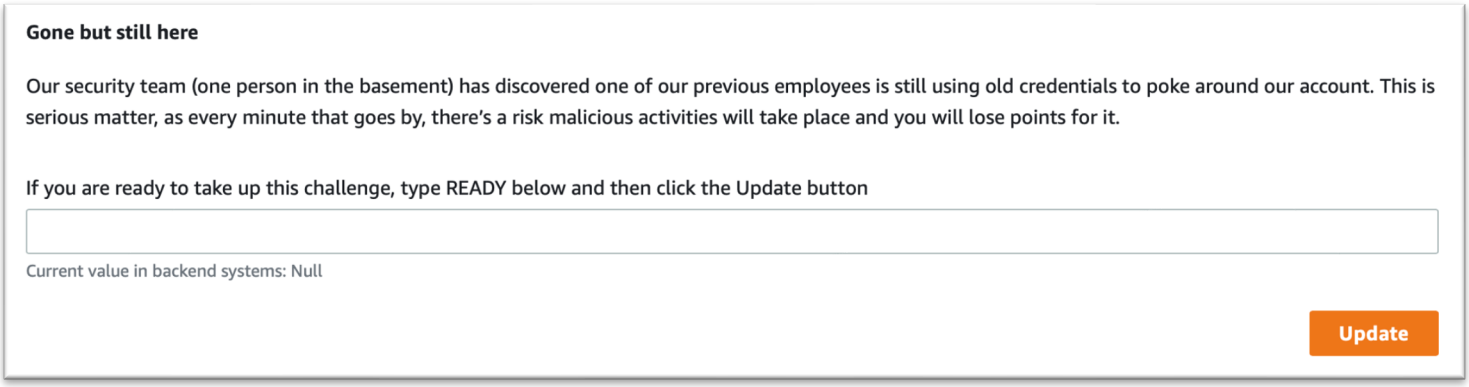



*Figure 10 - Before team's action*

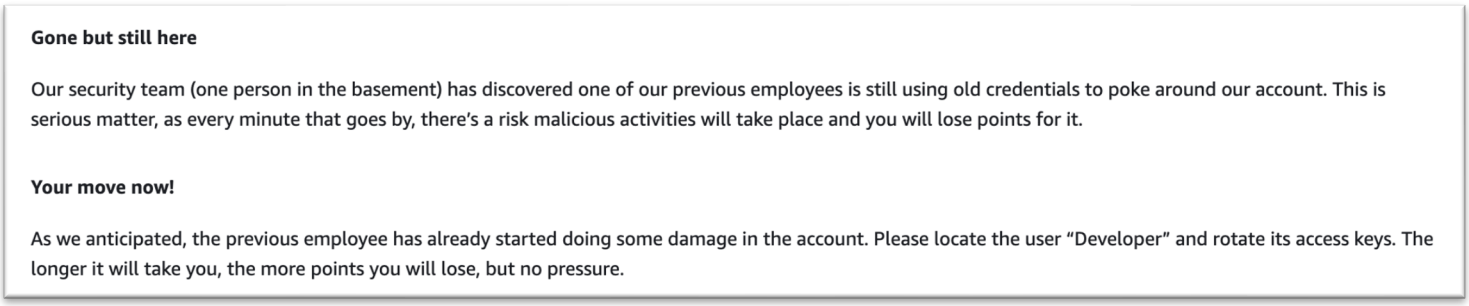



*Figure 11 - After team's action*

**[Negative Scoring on Chaos Event]** Points will start trending negative at 1-minute intervals at some (apparently random) point during the play-through. For instance, because something suddenly breaks, like a bad production rollout, a network change, or a malicious attack. Use proper wording to catch the team's attention that something needs immediate action. This pattern is usually pre-phased by the team having gained a good number of points until the chaos event. That is to avoid that the team goes minus-zero points as they struggle to put a



mitigation in place and experience anxiety or frustration. Use proper wording to catch the team's attention that something needs immediate action.

## Anti-cheat Mechanisms

The competitive integrity of an AWS GameDay Event is vital for participant experience, engagement, and learning. When building a Quest, make sure to think of ways teams might circumvent the actions the Quest design intends them to take. Following are few examples or scenarios to keep in mind. However, this is not an exhaustive list. Think strategically to avoid as many cheating paths as possible.

- Never trust any applications, IAM roles, or infrastructure in Team accounts. All scoring and evaluation should be via probe/pull from the central account, not push from the Team account.
- The team clicks a button twice in quick succession. As a result, points for a correct answer are awarded twice. A solution to this is to implement distributed locking (Figure 12) and use Boolean flags (Figure 4).

```
def save_team_data(team_data, quest_status_table):

    # Get the item's current version
    current_version = team_data["version"]

    # Increase the version number
    team_data["version"] += 1

    # Try updating the item, but only if it hasn't been updated by another function. Some possible
scenarios:
    # 1. Race condition between CHECK_TEAM_LAMBDA and UPDATE_LAMBDA with the former going first
    # 2. Race condition between CHECK_TEAM_LAMBDA and UPDATE_LAMBDA with the latter going first
    # 3. Race condition between two executions of UPDATE_LAMBDA due to rapid button clicks
    try:
        print(f"Storing team data back to DynamoDb: {json.dumps(team_data, default=str)}")
        dynamodb_response = quest_status_table.put_item(
            Item=team_data,
            ConditionExpression=Attr("version").eq(current_version)
        )
    except ClientError as err:
        if err.response["Error"]["Code"] == 'ConditionalCheckFailedException':
            raise ValueError("The item was updated by another function since this function started. Check
with the developer whether it is safe to ignore this error (the quest is not left in an inconsistent state
for the team)") from err
        else:
            raise err
    print(f"Persisted team data back to the quest team status table: {json.dumps(dynamodb_response)}")
```

*Figure 12 - Optimistic locking*



- Teams should not be able to guess the solution to a Quest challenge by discovering a cheating path. One example is identifying an IAM user/role by reviewing recent activities in the IAM console. The solution is to generated distractor users/roles and activities. Another example is to use mock user traffic to obscure malicious traffic
- Use IAM permissions to restrict console access where necessary on your buckets/objects. e.g., figuring out an object in a bucket could just be perused through the S3 console.
- When asking teams for ARNs or resource names, ensure they are from this team's AWS account. Teams should be prevented from referencing other Team's resources or resources external to the event (especially relevant for load balancers, CloudFront distributions, databases, and EC2 instances).



# Reference Quest Explained

The reference Quest is provided as an example of how to implement guidelines, best practices, and use cases. However, keep in mind that it only contains a subset of all the possible ways a Quest can be developed. Use the Guidelines and Best Practices section in this document for further guidance, and use best judgment whenever encountering a scenario that is not covered. If in doubt, contact the AWS GameDay team for additional guidance.

This section explains in more details the various aspects that comprise the Reference Quest.

## Tasks Summary

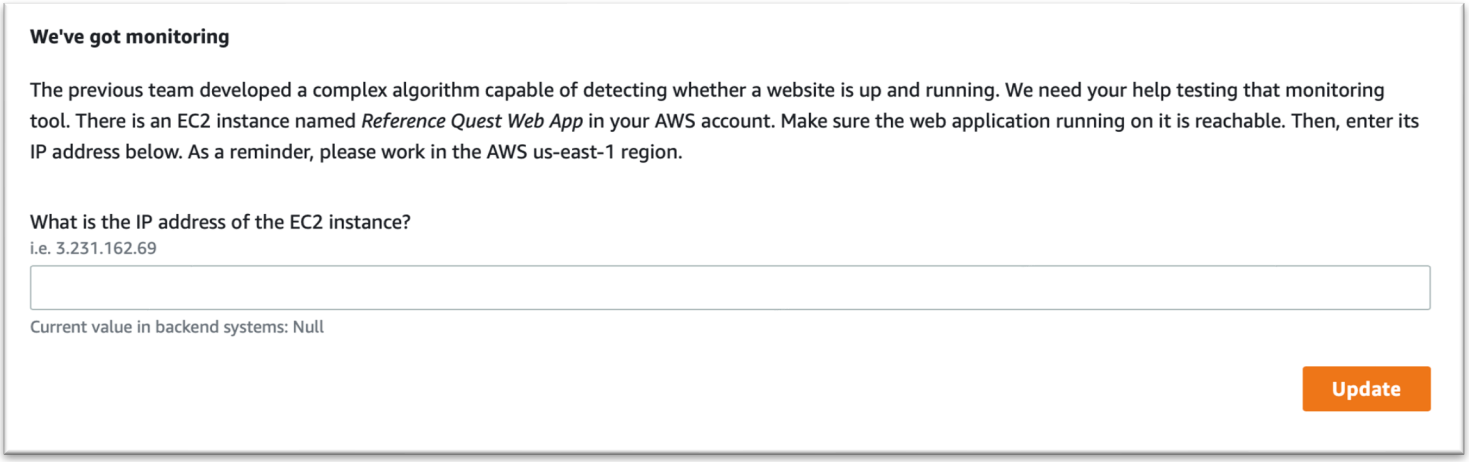
The Reference Quest comprises of 3 tasks that can be worked on in parallel, thus providing a non-linear play-through experience, and a fourth one that enables once all of the other 3 tasks are completed.

## Task 1 – We've got monitoring

This task implements the following use cases and design patterns among the others:

- User input
- Endpoint validation
- Continuous scoring
- Checkpoint scoring
- Assume team account's role to modify EC2 resources
- Chaos event

The task begins with *init_lambda.py* posting an output to the UI asking the team to provide the IP address of the EC2 instance that was provisioned in their account by the team enable template.



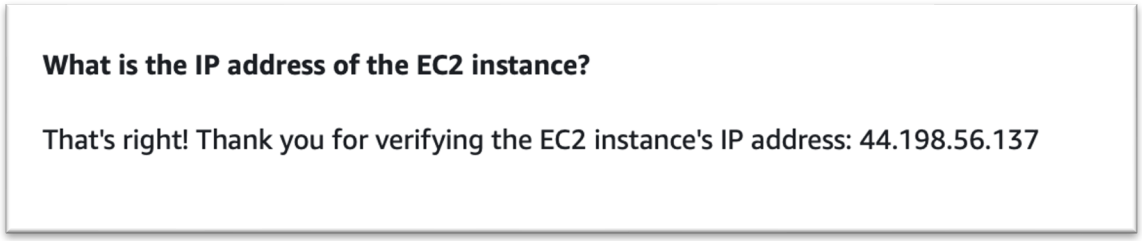

The team will navigate to the AWS EC2 console and retrieve the IP address for the EC2 instance named *Reference Quest Web App*.

After the team submits the IP address, *update_lambda.*py verifies its correctness and also does the followings:

- Saves the team status to DynamoDB right away to avoid the button is clicked twice potentially awarding double of the points:

```
        try:
            # First update DynamoDB to avoid race conditions, then do the rest on success
            dynamodb_utils.save_team_data(team_data, quest_team_status_table)
```

- Deletes a previous output message in case the team had provided a wrong answer before
- Transforms the input UI element into an output in order to show the team the history of what had been done. This is particularly useful when multiple people are working on the same Quest




- Awards points for the correct answer
- Sets the timer timestamp that will be used later on to calculate the chaos event start time:

```
        # Start chaos event timer
        team_data['monitoring-chaos-timer'] = int(datetime.now().timestamp())
```

In parallel, *check_team_lambda.py* monitors the task progress at 1-minute intervals doing the following:

- Waits for the team to provide the IP address before begin evaluating whether to trigger the chaos event:

```
    # Check if chaos has not started yet and the team has provided the EC2 IP address
    if not team_data['is-monitoring-chaos-started'] and team_data['is-ip-address']:
```

- Starts the chaos event either if the timer is up or task 2 and 3 have been already completed. This double condition is to avoid that a team has to potentially wait 10 minutes with nothing to do after completing all other tasks before the chaos event kicks in



```
        # Start chaos event if timer is up or task 2 and 3 are done
         if ((team_data['is-cloudshell-launched']    # Task 2
             and team_data['is-accesskey-rotated'])  # Task 3
             or is_chaos_timer_up(team_data['monitoring-chaos-timer'],int(CHAOS_TIMER_MINUTES))
             ):
```

- The chaos event start time is calculated as whether the time elapsed since the IP address was provided is greater than 10 minutes:

```
# Checks whether the chaos event timer is up by calculating the difference between the current time
# and the timer's start time plus the minutes to trigger the chaos event
def is_chaos_timer_up(timer_start_time, timer_minutes):

    # Timer start time
    start_time = datetime.fromtimestamp(timer_start_time)

    # Current time
    current_time = datetime.now()

    # Time difference
    time_diff = current_time - start_time

    # Time difference in minutes
    minutes = int(time_diff.total_seconds() / 60)

    if minutes >= timer_minutes:
        print(f"Chaos event timer is up: {minutes} minutes have elapsed")
        return True
    else:
        print(f"No time for chaos event yet: {timer_minutes - minutes} minutes left")

    return False
```

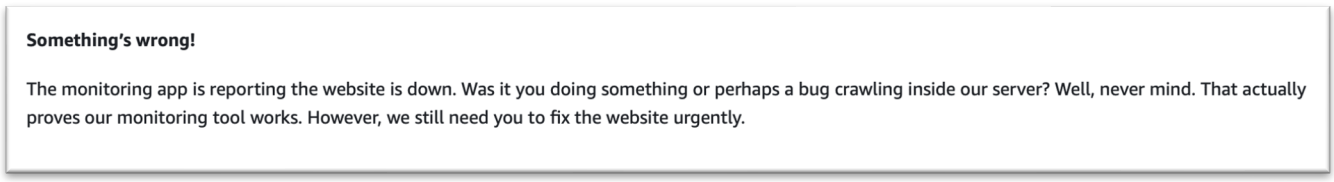
- Executes the chaos event by snapping the ingress rule on the EC2 instance's security group

```
    # Break network connectivity by removing ingress rules
    print(f"Removing ingress rules from Security Group {team_data['security-group']}")
    security_group = xa_session.resource('ec2').SecurityGroup(team_data['security-group'])
    response = security_group.revoke_ingress(
                IpPermissions=security_group.ip_permissions
              )
```

- Checks whether the web app running on EC2 is up or down, and awards or detracts points respectively

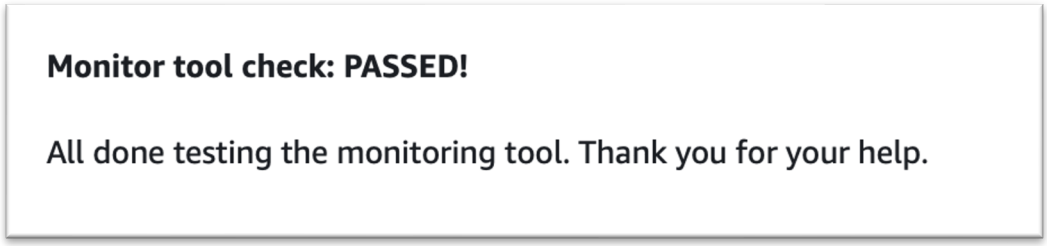


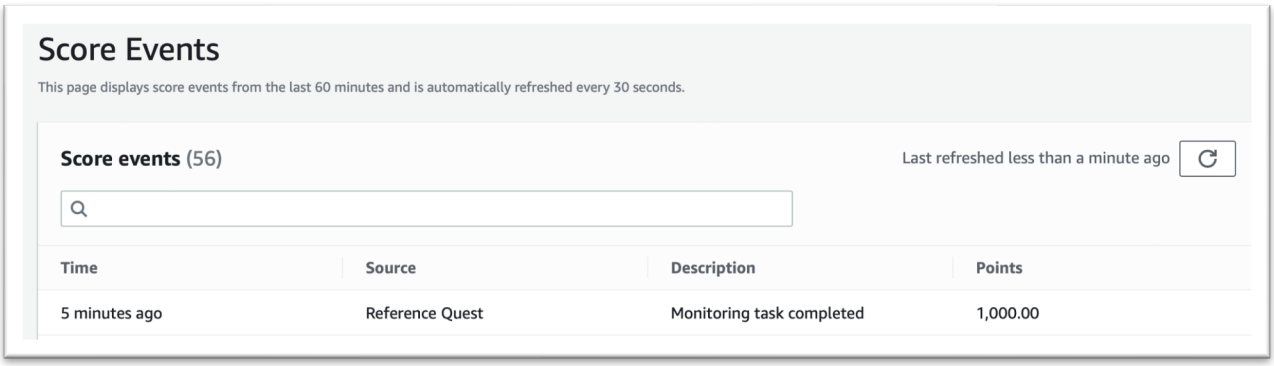
- Posts a warning message to the UI if the web app is down




- Keeps checking whether the team has fixed the EC2 instance's Security Group by adding an ingress rule on port 80 with source 0.0.0.0/0
- Completes the task and awards final points if the web app is up and the chaos event has been remediated

```
# Complete task if chaos event had started and web app is up
if team_data['is-monitoring-chaos-started'] and is_webapp_up:
```




## Task 2 – How is your curl?

This task implements the following use cases and design patterns among the others:

- Assume team account's role to evaluate logs in AWS CloudTrail
- Checkpoint scoring

The task begins with *init_lambda.py* posting an output to the UI asking the team to launch AWS CloudShell and run some cURL commands.



**How is your curl?**

As we need you to become familiar with basic tooling, please launch AWS CloudShell and use cURL to test the connection to the web app running on the EC2 instance named *Reference Quest Web App* in your AWS account .

In parallel, *check_team_lambda.py* monitors the task progress at 1-minute intervals doing the following:

- Assumes a role in the team account

```
# Establish cross-account session
print(f"Assuming Ops role for team {team_data['team-id']}")
xa_session = quests_api_client.assume_team_ops_role(team_data['team-id'])
```

- Looks up AWS CloudShell *CreateSession* events in CloudTrail

```
# Lookup CloudShell events in CloudTrail
cloudtrail_client = xa_session.client('cloudtrail')
quest_start = datetime.fromtimestamp(team_data['quest-start-time'])
cloudtrail_response = cloudtrail_client.lookup_events(
    LookupAttributes=[
        {
            'AttributeKey': 'EventName',
            'AttributeValue': 'CreateSession'
        }
    ],
    StartTime=quest_start
)
print(f"CloudTrail lookup result for team {team_data['team-id']}: {cloudtrail_response}")
```

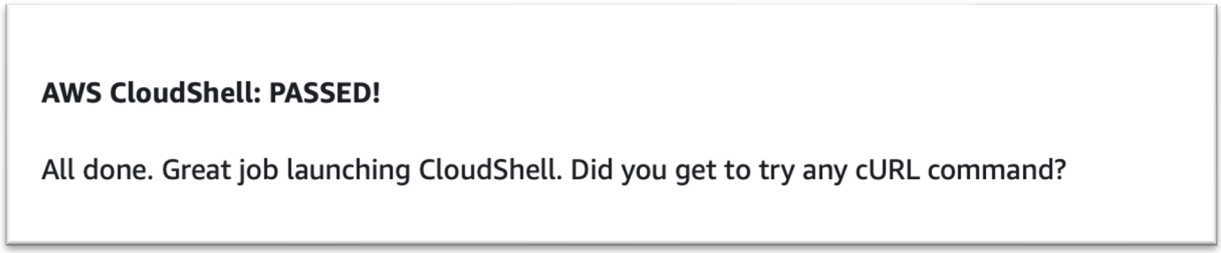
- Completes the task if the team has launched CloudShell and awards final points

```
# Complete task if CloudShell was launched
if len(cloudtrail_response['Events']) > 0:

    # Switch flag
    team_data['is-cloudshell-launched'] = True
```



```
        # Award final points
        quests_api_client.post_score_event(
            team_id=team_data["team-id"],
            quest_id=QUEST_ID,
            description=scoring_const.TASK2_COMPLETE_DESC,
            points=scoring_const.TASK2_COMPLETE_POINTS
        )
```

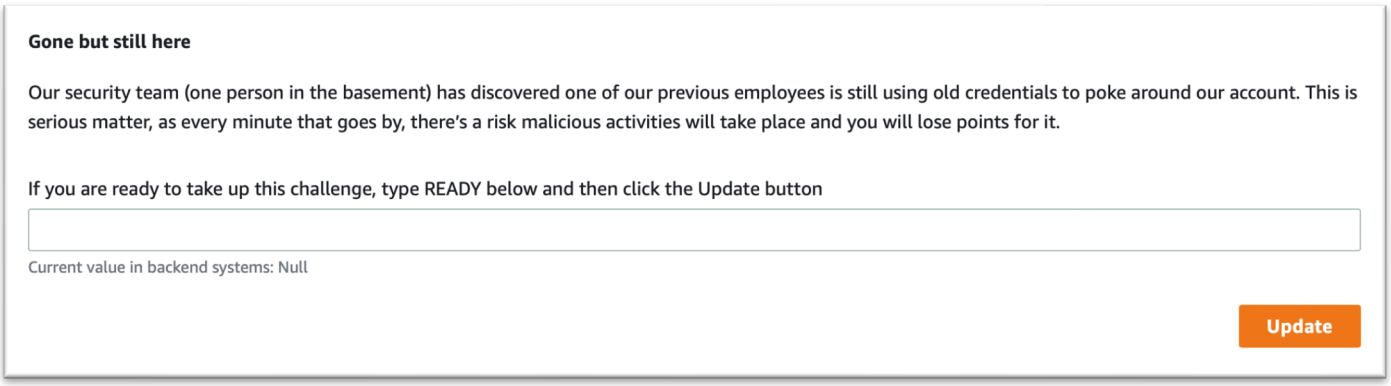



## Task 3 – Gone but still here

This task implements the following use cases and design patterns among the others:

- User input
- Continuous scoring
- Checkpoint scoring
- Assume team account's role to verify an IAM user

The task begins with *init_lambda.py* posting an output to the UI presenting a use case scenario and asking the team to take on the challenge by typing READY in a text box.

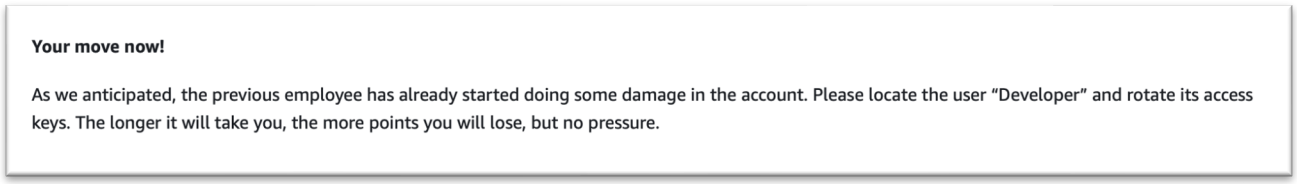



After the team types "READY" and clicks the Update, *update_lambda.*py does the followings:

- Saves the team status to DynamoDB right away to avoid the button is clicked twice potentially detracting double the points:
- Deletes the input, this time without replacing it with an output message since there wasn't really a meaningful question asked the team but just having them accept the challenge



- Post a message on the UI urging the team to take action




In parallel, *check_team_lambda.py* monitors the task progress at 1-minute intervals doing the following:

- Checks for the team accepting the challenge
- Assumes a role in the team account
- Lists the access key for the IAM user ‘ReferenceDeveloper’

```
# Check user's access key
iam_client = xa_session.client('iam')
keys = iam_client.list_access_keys(UserName='ReferenceDeveloper')
status = "Not found"
for key in keys['AccessKeyMetadata']:
    if (key['AccessKeyId'] == team_data['accesskey-value']):
        status = key['Status']
        print(f"Access key exists, checking if its active")
        break
```

- Checks whether the team has disactivated or deleted the access key
- Detract points if the access key is still active

```
if status == "Active":
    print(f"access key has not been deactivated")

    # Detract points
    quests_api_client.post_score_event(
        team_id=team_data["team-id"],
        quest_id=QUEST_ID,
        description=scoring_const.KEY_NOT_ROTATED_DESC,
        points=scoring_const.KEY_NOT_ROTATED_POINTS
    )
```

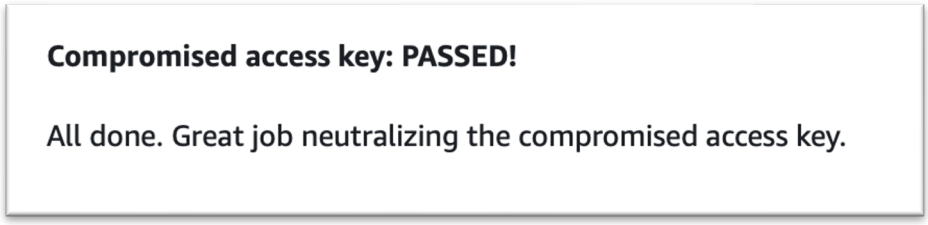
- Awards points and completes the task if the key was disactivated or deleted

```
elif status == "Inactive" or status == "Not found":

    print(f"Awarding points. Key has been deactivated or deleted")

    # Switch flag
    team_data['is-accesskey-rotated'] = True
```





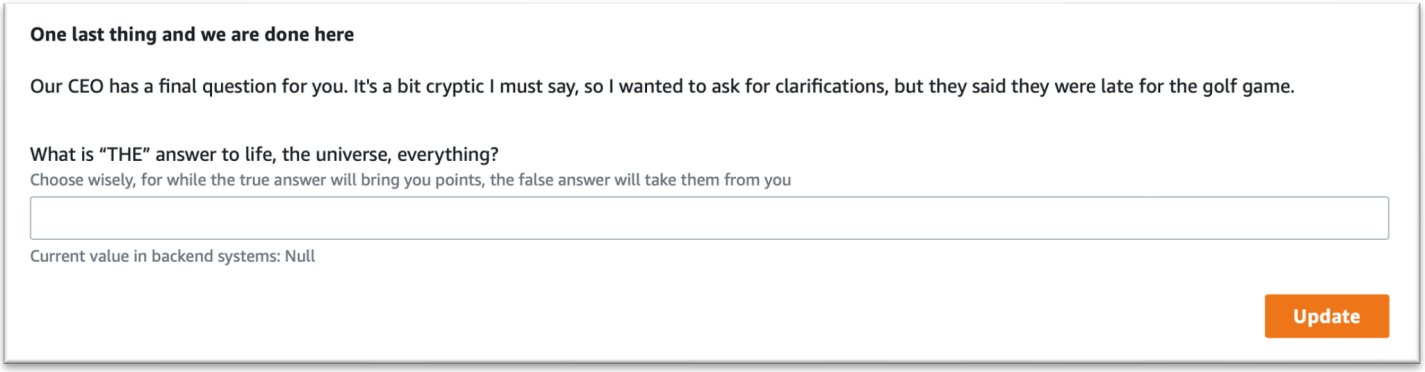

## Task 4 – One last thing and we are done here

This task implements the following use cases and design patterns among the others:

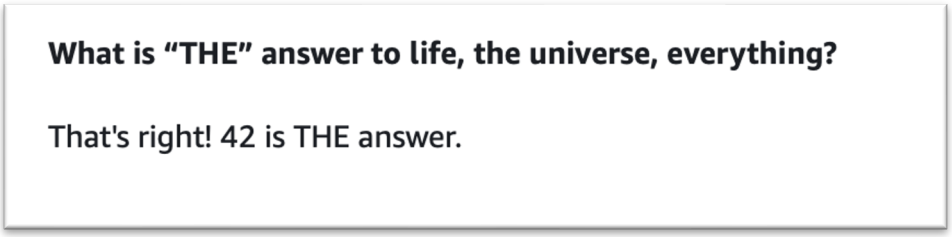
- User input
- Checkpoint scoring

Despite its simplicity, this task was added to demonstrate how to handle task dependencies. The task activation logic is in *check_team_lambda.py*, whereas *update_lambda.py* evaluates the team's answer and awards or detracts points based on its correctness.

Below is the output being displayed as the task is activated




Whereas following is the output on task completion




One important thing to notice in this task's code is how "42", "fortytwo", and "forty-two" are all evaluated as correct answers. As matter of fact, it is important not to penalize a team for providing a slightly incorrect answer.

## Quest completion

The *check_team_lambda.py* function checks whether the Quest has been completed at every execution. It evaluates whether the team has done with all tasks:



```
# Verify that all tasks have been successfully done and complete the quest if so
def check_and_complete_quest(quests_api_client, quest_id, team_data):

    # Check if everything is done
    if (team_data['is-monitoring-chaos-done']           # Task 1
        and team_data['is-cloudshell-launched']         # Task 2
        and team_data['is-accesskey-rotated']           # Task 3
        and team_data['is-answer-to-life-correct']):    # Task 4
```

If so, it does the followings:

- Awards Quest completion points
- Calculates and awards Quest bonus completion points
- Post a Quest completion message on the UI
- Marks the Quest as complete:

```
        # Complete quest
        quests_api_client.post_quest_complete(team_id=team_data['team-id'], quest_id=quest_id)
```

## Constant files

In order to improve the code readability and maintenance, some files of constants were created:

- *hint_const.py* contains values used when posting hints to the team via the *post_hint* call
- *output_const.py* contains values used when posting outputs to the team via the *post_output* call
- *input_const.py* contains values used when posting inputs to the team via the *post_input* call
- *scoring_const.py* contains values used when posting score events via the *post_score_event* call
- *quest_const.py* contains references to the possible states the event or Quest might be at a given time

## Negative Scoring Considerations

Both Task 1 and Task 3 will detract points from the team until some remediation actions are taken. As Quest developers, we need to be conscious of the fact that a team might not be actively working on the Quest, thus for them to start seeing points trending negative might be a cause of stress and frustration. A best practice is to start negative scoring only as a result of the team taking an action. For Task 1, the trigger is the team submitting the IP address. For Task 3, the trigger is the team explicitly stating they are ready to take on the challenge.

## Hints

The Quest provides hints for the team throughout the play-through to help them solve the different challenges. Whereas some hints are posted right away in *init_lambda.py* as the Quest starts, others are unlocked subsequently in response to some status change. For example, there is a



hint on Task 1 that suggests how to troubleshoot the broken web application. The hint is only posted after the chaos event starts.

# Troubleshooting

## Behavior Issues

| Issue | Reason | Resolution |
|---|---|---|
| “Error loading Quests” in UI | Bad data | Quest row in gdQuests DynamoDb table should have enabled=true |
| Changes not reflected in the UI after running package_quest_init.sh | User error | Package_quest_init.sh should only be used to initialize the Quest. Once the CodeCommit repo has been seeded from the S3 bucket, any changes of files present in CodeCommit should be uploaded to the CodeCommit repository directly. Running package_quest_init.sh will not alter those files anymore. |
| ./package_quest_init.sh: AWS-GameDay-Quests/quests/…/build/: Is a directory | Missing Env Var | Include the QUESTS_ARTIFACTS_ZIP in your package_quest_init.sh |

## Virtual Environment Setup

Commonly, virtual environment related issues can occur when packaging your Quest, e.g., with package_quest.sh

| Error | Error Reason | Resolution |
|---|---|---|
| command not found: enable-quest | Shell script commands not found | Deactivate virtual environment<br>1. Type deactivate into the CLI<br>2. Ctrl-D<br>PIPENV_VENV_IN_PROJECT="enabled";<br>pipenv install<br>Pipenv install<br>../../build/aws_gameday_quests.zip<br>or<br>source ./.venv/bin/activate |

## Enable-quest

When using the utilities included in the aws_gameday_quests package (e.g. *enable-quest)*, the following issues can occur

| Error | Error Reason | Resolution |
|---|---|---|
| Quest not IN_PROGRESS | Quest not IN PROGRESS in DynamoDB table gdQuestApi-quest-state | Click the activate button in the Quest UI for your Quest<br>or<br>Run enable-quest –quest-id [quest-id] –team-id [team-id] |



| | | |
|---|---|---|
| Exception: Quest taking >5 minutes to deploy. | CloudFormation Stacks are not deploying - Go to CloudFormation console | Click on the CloudFormation template that failed, go to the Events tab. Look for the resource that failed.<br>Manually test the CloudFormation templates end to end to capture a more descriptive error from CloudFormation |
| Object operation: The specified S3 Bucket does not exist | Bucket name cannot be found. | Ensure and S3 Bucket named your BUILD_QUEST_BUCKET_NAME variable exists and is in the same region as the Quest API CloudFormation Template. |

## CloudFormation Stack Errors

The following errors are possible when enabling or activating a Quest. The first CloudFormation stack to deploy is the gdQuests-QUESTID-Pipeline. It creates a CodePipeline pipeline which in turn, deploys other CloudFormation stacks, depending on which templates are configured.

| **Error** | **Error Reason** | **Resolution** |
|---|---|---|
| Lambda Creation errors:<br>Error occurred while GetObject.<br><br>S3 Error Code: NoSuchKey.<br><br>S3 Error Code: AccessDenied<br><br>S3 Error Message: The specified key does not exist | CloudFormation template is not able to find the file name in your S3 Bucket (S3 key) | See if the Lambda's definition in the CFN stack (S3Bucket + S3Key) matches an object in S3<br><br>Files uploaded in package_quest_init.sh are accessed through StaticAssetsBucket/StaticAssetsKeyPrefix CFN parameters.<br>Files uploaded in package_quest.sh using ${PIPELINE_BUCKET} are accessed throuh DeployAssetsBucket/DeployAssetsKeyPrefix CFN parameters |
| Quest not available, status: DISABLED, waiting 20 seconds…<br>Exception: Quest taking >5 minutes to deploy. Check the Quest CodePipeline and CloudFormation stack events. | Quest ID is not in the proper format | Ensure the Quest ID is a 27 char UUID |
| Resource creation cancelled<br><br>CloudFormation did not receive a response from your Custom Resource. Please check your logs for requestID[7a6ec30c-99f6-801a-5ea1-4519ddf9758e4]. If you are using the Python cfn-response module, you may need | Incorrect import statement or library version | To capture more descriptive error messages on the incorrect import, view the logs of the Lambda creating your custom resource in CloudWatch. The logs will only be present temporarily as CloudFormation will attempt to delete the resource along with the logs due the creation failure. |



| | | |
|---|---|---|
| to update your Lambda function code so that CloudFormation can attached the updated version. | | |
| [ERROR] Runtime.ImportModuleError: Unable to import module 'module_name': Unable to import required dependencies: | | |

## Custom Resources

| Error | Error Reason | Resolution |
|---|---|---|
| Reset-quest or manage-qdk hanging at Delete Complete. | A custom resource is stuck deleting. | https://aws.amazon.com/premiumsupport/knowledge-center/cloudformation-lambda-resource-delete/ |
| Resource creation cancelled | Resource is failing to create. | Preventing these errors from leaving CloudFormation stacks from hanging require the cfn_response module. An example can be found in the reference Quest under team_lambda_source/cfn_response.py and team_lambda_source/resource_lookup.py<br><br>To learn more about cfn_response module view the documentation here. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-lambda-function-code-cfnresponsemodule.html |
| CloudFormation did not receive a response from your Custom Resource. Please check your logs for requestId [307617a5-d1e7-4763-a21f-ba94ef8318d4]. If you are using the Python cfn-response module, you may need to update your Lambda function code so that CloudFormation can attach the updated version. | Resource is failing to delete and reset-Quest or manage-qdk commands are left hanging. | |

## Lambda Errors

| Error | Error Reason | Resolution |
|---|---|---|
| Source lambdas do not have any invocations in CloudWatch | 1. Lambda was not invoked in code | Reason 1 diagnosis – Check the CloudWatch logs of the Parent Lambda function to check if the Lambda was properly invoked. Match the error message to the messages in this table and continue troubleshooting |
| | 2. Source code did not successfully port into Lambda | Run a diagnosis on the Lambda that is not outputting the expected behavior by running a dummy Lambda test event. Match the error message to the messages in this table and continue troubleshooting. |



| | | |
|---|---|---|
| *{<br>"errorMessage": "Unable to import module 'lambda_name': No module named 'lambda_name'",<br>"errorType": "Runtime.ImportModuleError",<br>"stackTrace": []<br>}* | 1. CloudFormation lambda name mismatched | Verify the Handler value for your Lambda resource matches the name of the source code file in your central_lambda_source or team_lambda_source folder. |
| | 2. Compressed lambda files are wrapped in an extra folder | Check the file path in the build script go directly to the source code. If you are using the provided package_quest.sh, verify these lines of code align with your folder structure.<br><br>cd ${QUEST_ROOT_DIR}/central_lambda_source/.venv/lib/python3*/site-packages<br>zip -qr9 - . > ${QUEST_ROOT_DIR}/build/gdQuests-${BUILD_QUEST_NAME}-central-lambda-source.zip |
| File "/var/task/sns_lambda.py", line 13, in <module><br>ENVIRON_VAR_NAME = os.environ['ENVIRON_VAR_NAME']<br>File "/var/lang/lib/python3.7/os.py", line x, in __getitem__<br>raise KeyError(key) from None | CloudFormation Lambda Environment Variable names do not match variable name in source code. | Look at the source code and ensure the os.environ['string'] matches the CFn Lambda environment variable in the template. |

If you continue to experience issues, please report them to your AWS team.