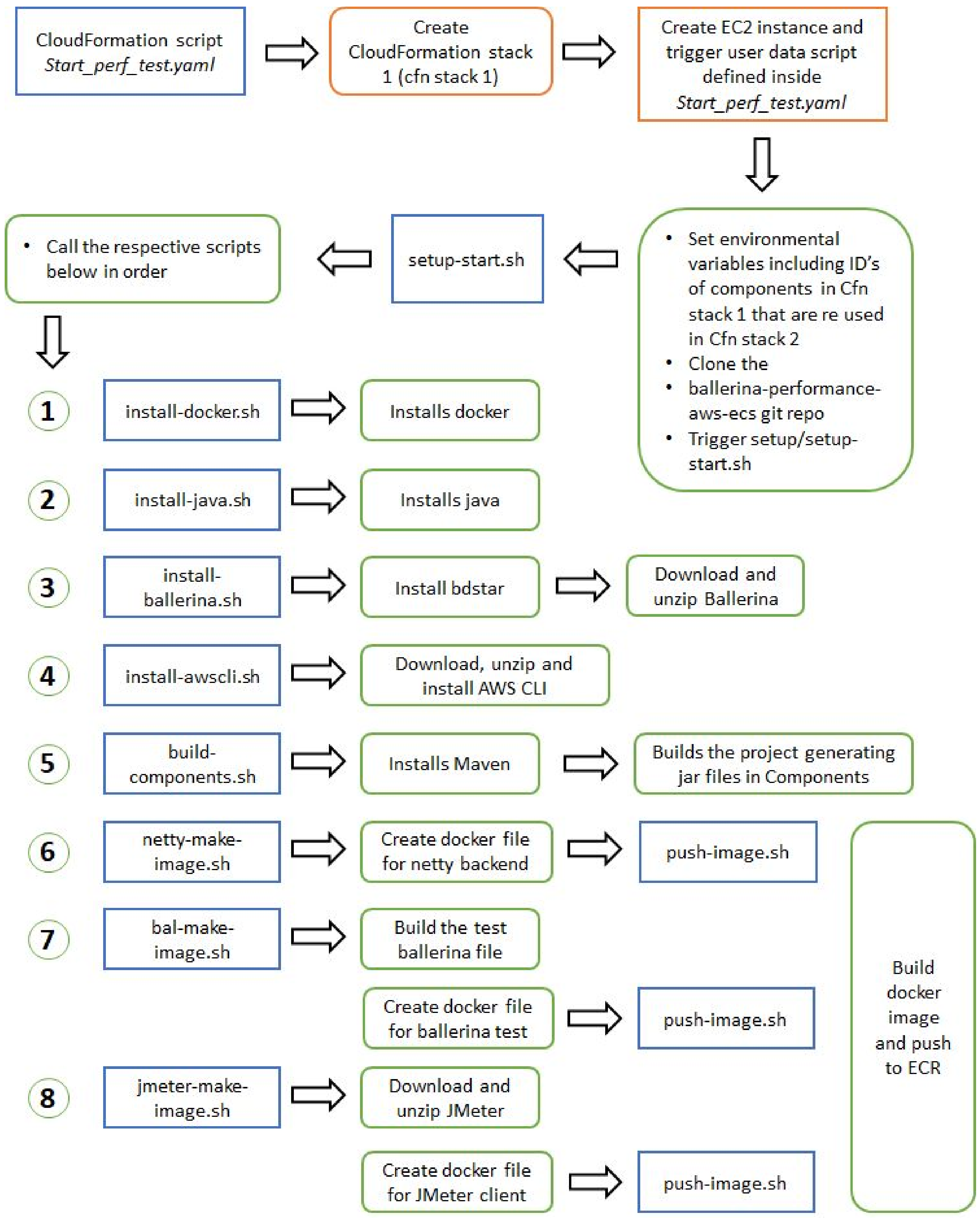
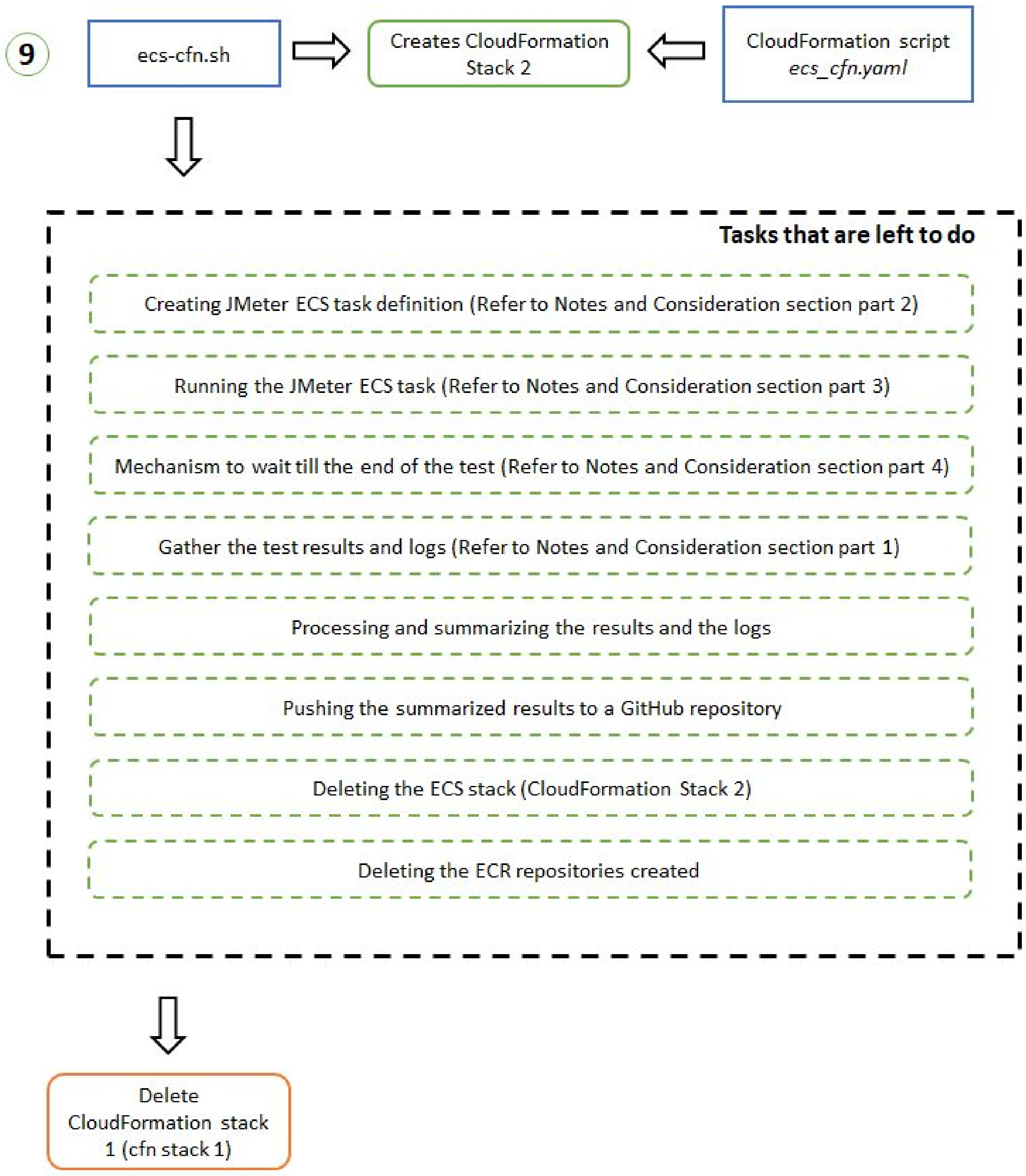
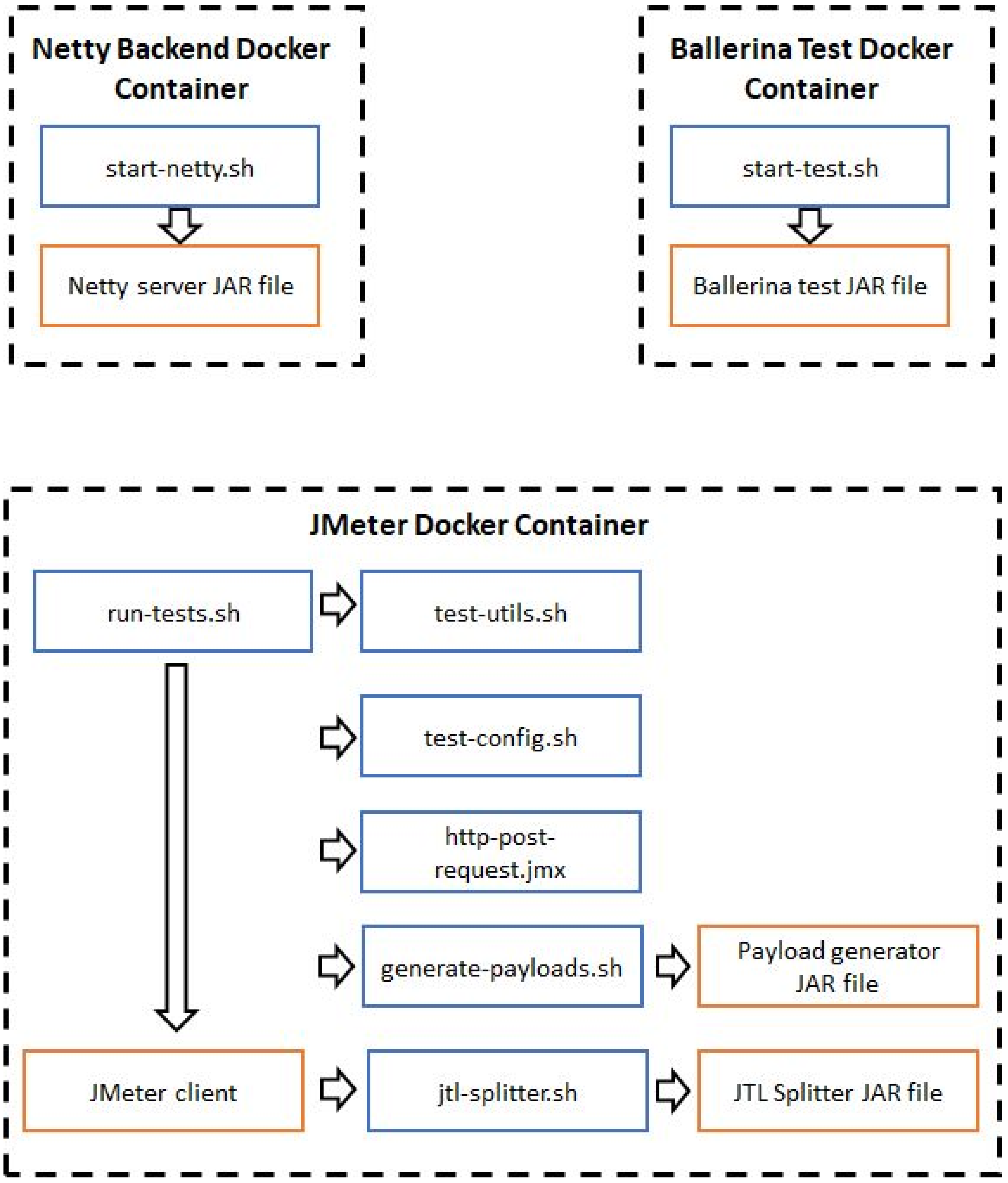
Workflow of the EC2 instance


CloudFormation script
Start_perf_test.yaml
Create CloudFormation stack 1 (cfn stack 1)
Create EC2 instance and trigger user data script defined inside *Start_perf_test.yaml*
• Set environmental variables including ID's of components in Cfn stack 1 that are re used in Cfn stack 2
• Clone the
• ballerina-performance-aws-ecs git repo
• Trigger setup/setup-start.sh
setup-start.sh
• Call the respective scripts below in order
1
install-docker.sh
Installs docker
2
install-java.sh
Installs java
3
install-ballerina.sh
Install bdstar
Download and unzip Ballerina
4
install-awscli.sh
Download, unzip and install AWS CLI
5
build-components.sh
Installs Maven
Builds the project generating jar files in Components
6
netty-make-image.sh
Create docker file for netty backend
push-image.sh
7
bal-make-image.sh
Build the test ballerina file
Create docker file for ballerina test
push-image.sh
8
jmeter-make-image.sh
Download and unzip JMeter
Create docker file for JMeter client
push-image.sh
Build docker image and push to ECR

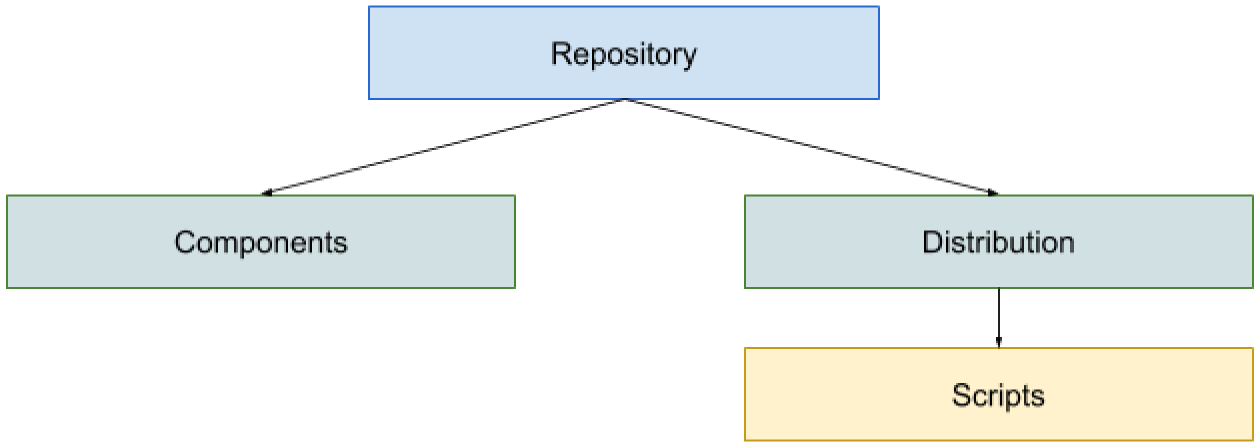
Continuation of the above diagram


9
ecs-cfn.sh
Creates CloudFormation Stack 2
CloudFormation script *ecs_cfn.yaml*
Tasks that are left to do
Creating JMeter ECS task definition (Refer to Notes and Consideration section part 2)
Running the JMeter ECS task (Refer to Notes and Consideration section part 3)
Mechanism to wait till the end of the test (Refer to Notes and Consideration section part 4)
Gather the test results and logs (Refer to Notes and Consideration section part 1)
Processing and summarizing the results and the logs
Pushing the summarized results to a GitHub repository
Deleting the ECS stack (CloudFormation Stack 2)
Deleting the ECR repositories created
Delete CloudFormation stack 1 (cfn stack 1)

Workflow inside the docker images



The repository structure and files



Components

- ***Jtl-splitter:*** Contains the Java source for the JTL splitter
- ***Netty-http-echo-service*:** Contains the Java source for the netty backend
- ***Payload-generator:*** Contains the Java source for the payload generator (it would generate payloads for the JMeter client to send based on different sizes inputted)

Distribution → Scripts

- ***Ballerina → tests***
    - ***H1c_h1c_passthrough.bal:*** Ballerina file for test server
- ***Ballerina***
    - ***Bal-make-image.sh (-t <test name>):***
        - Build the ballerina file (h1c_h1c_passthrough.bal)
        - Create docker file for ballerina test
        - Call *docker/push-image.sh*
    - ***Install-ballerina.sh:***
        - Download the Ballerina zip from the link
        - Install bdstar on the system
        - Use bdstar to unzip the Ballerina
    - ***Start-test.sh ([-n <Address of netty host>] [-t <Name of the test>])***
        - This file is copied to and runs inside the docker image (created in *Bal-make-image.sh*)
        - This is the entrypoint for the image
        - It triggers the the jar that was built for the Ballerina test
    - ***Test-config.sh***
        - This is used by the scripts used inside JMeter. It is copied to the docker image when making the JMeter image.

    - It contains the JMeter configurations for all the Ballerina tests that are available.
- ***Cloudformation → templates***
  - ***Start_perf_test.yaml:***
    - The first cloudformation script that is used (more details in **Cloudformation scripts** section of this document).
  - ***Ecs_cfn.yaml:***
    - The second cloudformation script used (more details in **Cloudformation scripts** section of this document).
- ***Cloudformation***
  - ***Ecs-cfn.sh:***
    - Used by EC2 to set up the second cloudformation stack using *Ecs_cfn.yaml*
- ***Docker***
  - ***Install-docker.sh***
    - Used to install docker in the EC2
  - ***Push-image.sh ([-d <docker_file_location>] [-r <Name for ecr repo>] [-i <Name for image>] [-t <Tag for image>])***
    - This is commonly used to build and push Netty, Ballerina and JMeter docker images to ECR.
    - Uses the dockerfile to build docker image with image name and tag specified.
    - Logs into the AWS ECR
    - Checks if the repository already exists. If it does, it deletes that repository.
    - Create a new repository.
    - Push the built docker image.
- ***Java***
  - ***Install-java.sh***
    - Used to install Java in the EC2
- ***Jmeter***
  - ***Jmeter-make-image.sh***
    - Download and extract Jmeter
    - Create JMeter docker file
    - Call *docker/push-image.sh*
  - ***Run-tests.sh***
    - This is the main script used inside the JMeter docker image to run the JMeter tests.
    - Refer to **JMeter test** section of this document for more information
  - ***Generate-payloads.sh ([-p <payload_type>] [-s <payload_size>])***
    - This script is used inside the JMeter docker image to use the payload generator jar to generate a payload.
  - ***Jtl-splitter.sh ([-m <heap_size>] [-h] -- [jtl_splitter_flags])***

            - Used inside the JMeter docker image to use the JTL splitter jar to split the JTL files
        - ***Test-utils.sh***
            - Various utility functions used by *Run-tests.sh*
        - ***Http-post-request.jmx***
            - JMX file used for HTTP POST request in JMeter
    - ***Netty***
        - ***Netty-make-image.sh***
            - Creates dockerfile for the netty background
            - Call *docker/push-image.sh*
        - ***Start-netty.sh***
            - Used inside the netty backend docker image to start the netty server (as an entrypoint)
    - ***Setup***
        - ***Setup-start.sh***
            - Starting point for the EC2 setup
            - Responsible for calling all the other responsible scripts in order
        - ***Install-awscli.sh***
            - Script for installing the AWS CLI
        - ***Build-components.sh***
            - Used to install maven and build the **Components** (jtl-splitter, netty and payload generator)

CloudFormation Stack 1

Parameters:

- Email Address: For tagging
- Key Name: To SSH into the EC2 instance
- Github URL: To clone the “ballerina-performance-aws-ecs” repository
- Ballerina Zip URL
- JMeterOptions: Given as a string. Refer to the **JMeter test** section for details.
- Memory to be used for the Ballerina test ECS task
- CPU to be used for the Ballerina test ECS task
- Instance type for the EC2

Resources:

- IAM role (EC2ResourceAccessRole) for the EC2 to carry out tasks. Permissions given:
    - ECR Full access: To create, push and delete ECR repositories
    - CloudFormation Full access: To create the CloudFormation stack 2
    - IAM Full access: Needed when creating Cloudformation ECS stack when creating the new ECS task execution role needed.

    - Elastic Load balancer Full Access: To create and link a new load balancer in the CloudFormation stack 2
    - Amazon ECS full access: Used to access ECS functions by CloudFormation Stack 2
- Instance profile with the above role for EC2
- VPC
- Public subnet for the EC2 instance and the JMeter ECS task
- Internet gateway for the Public Subnet
- Attach gateway to attack Internet gateway to the VPC
- Public Route Table for the Public Subnet
- Route entry in the Public Route Table
- Public subnet route table association
- Private Subnet for the Docker container
- Nat gateway for the private subnet
- Private route table for the Private subnet
- Private route entry to the table
- Private subnet route table association
- Security group
- EFS file system
- Mount target for the above File system and Public subnet
- EC2 Instance

CloudFormation Stack 2

This stack is created by the EC2 instance created in CloudFormation Stack 1.

Resources and parameters used from stack 1:
- Email address
- Private subnet
- Security group
- VPC
- Memory for the Ballerina Task
- CPU for the Ballerina Task

Other parameters in addition to the above:
- Netty image ECR link
- Ballerina Test image ECR link

Resources:
- IAM role (ECSTaskExecutionRole) to allow the ECS tasks to pull images from ECR and create logs.

- ECS cluster
- Network load balancer
- Netty backend Task definition
- Netty service using the above task definition
- Target group to point to the Netty task
- Load balancer listener to listen to a particular port and forward to the above target group
- Ballerina test Task definition
- Ballerina service using the above task definition
- Target group to point to the Ballerina test task
- Load balancer listener to listen to a particular port and forward to the above target group

JMeter test

The JMeter test is started by the ***run-tests.sh*** script inside the JMeter docker container. The following options are available.

-m: Application heap memory sizes. You can give multiple options to specify multiple heap memory sizes. Allowed suffixes: M, G.
-u: Concurrent Users to test. You can give multiple options to specify multiple users.
-b: Message sizes in bytes. You can give multiple options to specify multiple message sizes.
-d: Test Duration in seconds. Default: 900 seconds
-w: Warm-up time in seconds. Default: 300 seconds
-k: Heap Size of JMeter Client. Allowed suffixes: M, G. Default: 2G
-l: Heap Size of Netty Service. Allowed suffixes: M, G. Default: 4G
-i: Scenario name to be included. You can give multiple options to filter scenarios.
-e: Scenario name to be excluded. You can give multiple options to filter scenarios.
-t: Estimate time without executing tests.
-p: Estimated processing time in between tests in seconds. Default: 60 Seconds
-h: Display this help and exit.

When creating the CloudFormation Stack 1, the above parameters are taken as a string.

Example: *-u 100 -b 512 1024 2048 -d 600 -w 150 -i h1c*

The ***test-config.sh*** can be used to add new tests and change their configurations.
All the logs and results will be generated inside a folder named *results*.

Notes and Considerations

1. There is a concern to how the results and logs generated by the JMeter ECS task and the other tasks are to be collected. There are two suggested options to take.
    a. Using an AWS EFS file system. The EFS can be mounted onto the EC2 and the ECS tasks (JMeter task primarily) and they can write the results and logs directly to the file system. After the tests are done the EC2 can gather the results directly from the mounted EFS, process them, summarize them and push to a repository. This approach was considered so far in the work in progress application.
    b. The ECS tasks push the results and logs to an external repository (an S3 bucket or a Git repo) and the EC2 can pull them and process/summarize them.
    c. The ECS tasks themselves process the results and push them directly.

2. The JMeter ECS task definition was not included in the CloudFormation Stack 2 as the EFS approach was considered so far, and CloudFormation currently does not support mounting EFS to the task. There are two approaches to handle this:
    a. Using the AWS CLI on the EC2 instance to directly create the task definition along with the EFS mount.
    b. Ditching the EFS approach altogether, which would enable to define the task in the CloudFormation Stack 2 itself.

3. CloudFormation also does not support running the task. It only allows defining a service using the task definition. The issue with this for the JMeter task is that we only require the JMeter task once. If run as a service, once the task stops running, another JMeter task would be automatically generated. There are two approaches to handle this:
    a. Using the AWS CLI on the EC2 instance to run the task using the task definition ID.
    b. Adding a block to the JMeter task container, such that it waits instead of exiting the container once the tests are completed. In this scenario the JMeter Service can be defined in the CloudFormation Stack.

4. If the EC2 is expected to gather and process the JMeter results and logs (unless approach 1.c above is taken), there must be a mechanism to make the EC2 instance wait until the test is over. The two approaches that can be taken:
    a. Set a predefined/ calculated waiting time period and direct the EC2 to wait.
    b. Set up a listener in the EC2, and send some signal from the JMeter task once the JMeter tests are finished.

5. For the IAM role (EC2ResourceAccessRole) made for the EC2 in CloudFormation Stack 1, full access was given for resources such as ECS, IAM etc. Limiting these only to the actual permission that are used is recommended.

6. There are no checks included in the EC2 setup scripts to ensure that the dependencies such as Java, Ballerina, Maven etc have been installed properly.

7. The *StatCalculatorTest* in the JTL splitter code (taken from the Performance commons repository) takes a large amount of time and processing to finish. Hence, it was disabled in the module Maven POM.

8. Only the Memory and CPU allocations for the Ballerina Test task were parameterized, while the allocations for the Netty Backend was hard coded.

9. Some of the resources and parameters in CloudFormation Stack 1 is used by CloudFormation Stack 2 as well. In order to pass them (parameters and ID's of the resources that are allocated dynamically), they were exported as environmental variables in the EC2 instance user data in the CloudFormation Stack 1 script.