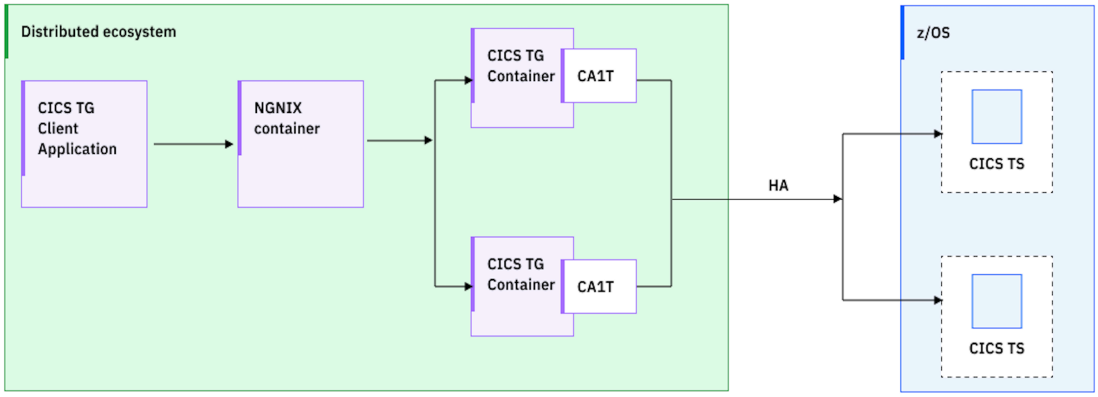
Introduction

CICS TG load balancing is of utmost importance in preserving system availability through the equitable distribution of transaction workload among various components. Specifically, TCP/IP load balancing focuses on guaranteeing the high availability of the CICS transaction Gateway daemon. This is accomplished by effectively distributing client connections across multiple Gateway daemons.

In this project, we will illustrate the steps to configure multiple CICS TG containers with high availability. Additionally, we will utilise an 'NGINX' container for load balancing of TCP/IP requests originating from the CICS TG client application, which will be routed to both CICS TG and subsequently to CICS TS.



CA1T offers sample exits that facilitate the rapid and efficient setup of a high availability infrastructure between CICS TG and CICS. These exits are designed to work with rules defined in a simple, text-based configuration file. The CA1T configuration file acts as a central repository where administrators can define and customize various rules related to high availability.

Prerequisites

Listed below are the prerequisites.

- CICS Transaction Gateway 9.3 container image
- Get the CICS TG client.jar (ctgclient.jar) and samples jar(ctgsamples.jar) file from CICSTG SDK. Download the CICS TG SDK here.
- Supported Docker version 23.0.5

Configuration

1. Configuration for the CICS TG Client Application

The provided Dockerfile, which is located in the javaApp directory, contains the necessary configuration for building the CICS TG client Application files, enabling the execution of the sample application.

In the Dockerfile, the image is constructed using the openjdk:8 parent image sourced from Docker Hub. Subsequently, the required CICS TG files essential for running the sample application are copied. The working directory is then set, and a Docker entry point is specified for the container upon start up.

To build the container successfully, ensure that you have downloaded the CICS TG client library (ctgclient.jar) and the sample jar from the CICS TG SDK. Place these files in the current directory before proceeding with the container build process. The Dockerfile will then copy the necessary files from the current directory into the container for executing the sample application. Dockerfile for the CICS TG client container

```
FROM openjdk:8
ADD ctgclient.jar /opt/ctgclient.jar
ADD ctgsamples.jar /opt/ctgsamples.jar
WORKDIR /opt/
CMD [/bin/bash]
```

To build the Docker image from the Dockerfile, follow these steps:

Use the `docker build` command to build the image. Provide a name for the image using the -t flag followed by the desired image name. The period . at the end indicates that the build context is the current directory.

```
docker build -t openjdk-tg.
```

2. Configuration for the ngnix container

NGINX is an open-source web server software commonly used for reverse proxy, load balancing, and caching purposes. The provided Dockerfile contains the necessary configuration to build a Docker image based on the NGINX base image. It includes an 'NGINX' configuration file that allows you to define how NGINX handles requests for server resources.

To build the nginx Docker image from the Dockerfile, follow these steps:

```
docker build -t nginx.
```

Building the Docker image from the provided Dockerfile will create a containerized NGINX instance with the specified configuration, allowing you to deploy and utilize it for load balancing purposes between two CICS TG containers. The load balancing algorithm used in this configuration is a round robin one, meaning that incoming requests will be evenly distributed in a sequential order to each CICS TG container on a rotation basis.

Once the Docker image is built and the container is deployed, NGINX will serve as a load balancer, forwarding incoming requests to the CICS TG containers using the round robin algorithm. This setup helps distribute the workload effectively and ensures better resource utilization across the CICS TG containers.

The provided NGINX configuration demonstrates how to configure NGINX as a TCP load balancer using the Round Robin algorithm. The stream context is used to define the TCP load balancing configuration.

```
stream {
    upstream tcp_backend {
        zone tcp_backend 64k;
        server cicstg1:2007 weight=5;
        server cicstg2:2007 weight=5;
    }
    server {
        listen 2007;
        proxy_pass tcp_backend;
    }

}
```

Within the stream context, an upstream block named tcp_backend is defined. This block specifies the servers that NGINX will load balance traffic to. In this case, cicstg1:2007 and cicstg2:2007 are specified as the servers with a weight of 5 each. The weight determines the proportion of traffic that each server receives in comparison to others.

The server block within the stream context specifies the listening port, in this case, port 2007. The proxy_pass directive is used to pass the incoming TCP traffic to the tcp_backend upstream group, which will handle the load balancing and distribute the traffic to the configured servers using the Round Robin algorithm.

With this configuration, NGINX will listen on port 2007 and forward incoming TCP traffic to the servers cicstg1:2007 and cicstg2:2007 in a round-robin manner, effectively load balancing the traffic between the two CICS TG containers.

3. Configuration for the CICS TG container

Download the CICS Transaction Gateway image from IBM Passport Advantage based on the part number 'n' to a temporary directory, where 'n' is the part number listed in Tables.

After downloading the CICS TG container image, load the image on docker.

The IBM® CICS TG container available in tar.gz format is loaded through docker load command.

```
docker load -i ibm-cicstg-container-linux-x86.tar.gz
```

Upon successful completion of docker load, use the docker images command to view the image.

```
docker images
```

Download the 'cicstg1' and 'cicstg2' directories and place them in your current directory. The configuration files, such as 'ctg.ini' and 'ctgd.conf', will be utilized to initiate the gateway upon container startup. Two sample CICS servers (CICSSRV1 and CICSSRV2) are provided with the configuration file 'ctg.ini'.

4. Setup the docker compose file for multiple containers

Download the content of the 'docker-compose.yaml' file that can be used to start CICS TG and nginx multiple containers.

```
version: '3.3'
services:
  cicstg1:
        image: localhost/ibm-cicstg-container-linux-x86:9.3
        container_name: cisctgcontainer1
        privileged: true
        restart: always
        environment:
          - LICENSE=accept
        volumes:
          - ./cicstg1:/var/cicscli
        ports:
          - "2009:2007"
  cicstg2:
```

```
        image: localhost/ibm-cicstg-container-linux-x86:9.3
        container_name: cisctgcontainer2
        privileged: true
        restart: always
        environment:
          - LICENSE=accept
        volumes:
          - ./cicstg2:/var/cicscli
        ports:
          - "2010:2007"
 javaApp:
        image: openjdk-tg:latest
        container_name: java-TG-App
        privileged: true
        restart: always
        depends_on:
          - nginx

nginx:
        image: nginx:latest
        container_name: nginx-route
        privileged: true
        restart: always
        depends_on:
        -  cicstg1
        -  cicstg2
        ports:
        - "2007:2007"
```

The provided Docker Compose file defines our multi-container application.

The version of the Docker Compose file used is 3.3, which is supported for specific Docker releases.

This docker-compose define a total of four services: cicstg1, cicstg2, javaApp, and nginx using this Docker Compose file to spin up multiple containers simultaneously or tear them down.

Create two separate services for CICS TG, cicstg1 and cicstg2, with the host port set to 2009 and 2010 respectively. Inside the containers, both services will be running on port 2007. Set the necessary environment variables and use volume mapping to persist container files on the host. Volume mapping can also facilitate the updating of configuration files within the containers. You can have different configurations for CICS TG by mounting different folders.

Create the services for the CICS TG client application using the `openjdk-tg` based on the provided Dockerfile. This service depends on the nginx service, ensuring that javaApp starts before NGINX.

The NGINX service depends on the cicstg1 and cicstg2 services, meaning that the NGINX service will start after the CICS TG services. The round robin algorithm will be employed to route requests from 'nginx-route' to either 'cicstgcontainer1' or 'cicstgcontainer2'.

5. Run the Docker-Compose instance

The Docker Compose file defines the services required for the multi-container setup, including CICS TG containers, a Java application container, and an NGINX container acting as a reverse proxy. Each service has its specific configurations, such as image, container name, restart policy, environment variables, volume mounts, and port mappings.

By running docker-compose up, Docker Compose will orchestrate the creation and configuration of these containers based on the provided Docker Compose file.

```
docker compose up
```

This will start instances of the CICS TG Client Application, NGINX, and the CICS TG containers. The CICS TG client application will utilize the NGINX port sharing option to share the port between the CICS TG containers and execute the EC01 program on CICS TS. You can run the sample from the 'openjdk-tg' container and route the requests to the defined CICS TG containers.

How to run the sample

After building the Docker images and running the containers using the

`docker-compose up` command, you can check if the containers started successfully by using the `docker ps` command. This command will list the running containers.

To access the CICS TG client application container, you can use the container ID that is generated when the container starts.

```
docker exec -it container_id /bin/bash.
```

This command will redirect inside the container to the /opt directory where all the volume mapped files from the host files system are present.

Add the ctgclient and the ctgsamples.jar to the CLASSPATH.

```
export
CLASSPATH=$CLASSPATH:$PWD/ctgclient.jar:$PWD/ctgsamples.jar
```

Running the sample:

```
java      com.ibm.ctg.samples.eci.EciB2      jgate=nginx-route
jgateport=2007 server=CICSSRV1 prog0=EC01 COMMAREAlength=18
```

To run the ECI application, we are utilizing the NGINX container with the jgate parameter 'jgate=nginx-route'. The round robin algorithm implemented in NGINX will be employed to route requests from 'nginx-route' to either 'cicstgcontainer1' or 'cicstgcontainer2'.

Configuring Ca1t for High availability and workload balancing between CICS TG and CICS servers

Use the Workload Manager (CA1T) to dynamically select servers and effectively balance the workload in your system while performing request validation. The Workload Manager is exclusively used for connections to IPIC CICS servers.

To ensure that the ca1t.ini configuration is used by the gateway daemon configuration file, use the ca1t.ini sample file and place it in the cicstg1 and cicstg2 directories. These directories should be mounted when the container starts.

To define custom server rules for remapping and retrying failed requests based on workload balancing policies, configure the sample HA configuration file. In this case, we will use the round robin policy, which distributes requests in a round-robin manner, ensuring an even distribution across the servers.

For more information about creating HA configuration files, see High Availability(HA) and Request validation configuration.

In our HA configuration file, SERVER1 is set to CICSSRV1 and CICSSRV2.

The location of the ca1t.jar has been included in the CLASSPATH configuration for CA1T with Gateway. This jar can be found at /opt/ibm/cicstg/classes/ca1t.jar. The ‘ctgd.conf’ file in cicstg1 and cicstg2 has been updated to add the CLASSPATH entry for ca1t.jar.

```
export CLASSPATH=/opt/ibm/cicstg/classes/ca1t.jar
```

Specify the location of your HA configuration file for the Gateway by using the variable CTG_HACONFIG. This variable is defined in the ‘ctgd.conf’ file of your setups.

```
export CTG_HACONFIG="/var/cicscli/ca1t.ini"
```

To Ensure that the gateway has the read access for the ca1t configuration file, change the permission of the file in the cicstg1 and cicstg2 directory.

```
chmod 640 /var/cicscli/ca1t.ini
```

To specify the round-robin workload management policy, update the Gateway daemon configuration file (ctg.ini) in cicstg1 and cicstg2. In the ctg.ini file, specify the usage of the RoundRobinExit by setting cicsrequestexit to RoundRobinExit, which enables the round-robin policy.

```
cicsrequestexit=com.ibm.ctg.ca1t.ha.RoundRobinExit
```

To enable the use of a log file for the ca1t configurations, add the CTGD_PARAM variable in the gateway daemon configuration file (ctgd.conf). The following configuration in the ctgd.conf file will provide a full debug trace for ca1t Level 3 logging, and the trace file will be located in the directory /var/cicscli/ca1t.out.

```
CTGD_PARAMS="-j-Xmx2048M -j-Xms2048M -j-
Dcom.ibm.ctg.ca1t.haexit.out=/var/cicscli/ca1t.out -j-
Dcom.ibm.ctg.ca1t.haexit.ca1t=3"
```

For more details on configuring refer the page.

How to run the Sample with CA1T configuration

To access the CICS TG client application container, you can use the container ID that is generated when the container starts.

```
docker exec -it container_id /bin/bash.
```

This command will redirect inside the container to the /opt directory where all the volume mapped files from the host files system are present.

Add the ctgclient and the ctgsamples.jar to the CLASSPATH.

```
export
CLASSPATH=$CLASSPATH:$PWD/ctgclient.jar:$PWD/ctgsamples.jar
```

Running the sample:-

```
java      com.ibm.ctg.samples.eci.EciB2      jgate=nginx-route
jgateport=2007 server= SERVER1 prog0=EC01 COMMAREAlength=18
```

To run the ECI application, we are utilizing the NGINX container with the jgate parameter 'jgate=nginx-route'. The Round Robin algorithm implemented in NGINX

will be employed to route requests from 'nginx-route' to either 'cicstgcontainer1' or 'cicstgcontainer2'. The server parameter will be used with SERVER1, and the request will be routed in a round-robin manner to either CICSSRV1 or CICSSRV2, which are CICS Servers.

Conclusion

The setup mentioned above demonstrates High Availability (HA) on CICS TG containers at both the client level and the CICS TG level. This HA implementation spans across both the client-side and CICS TG-side components, ensuring a resilient and highly available environment for CICS TG operations.