# Hermes 2.0 Hack-a-thon
# Gameplay Manual

November 1-3, 2013

# Table of Contents

# Overview

## Introduction

In today's increasingly global and fast-moving trading world, banks need to react ever-faster to changing business opportunities, and deploy technology to address client demand whenever and wherever needed. This challenge will give you a practical introduction and insight into the challenges of trading infrastructure, and will present you with an opportunity to showcase your critical thinking, teamwork, and programming skills.

In previous deployment models at Morgan Stanley, applications have been deployed on a fixed number of physical servers, in specific locations (e.g. New York City). And while this has traditionally worked well for local clients of the application, those in different regions (e.g. Europe, Asia-Pacific) receive a poorer service experience. Worse still, if the demand increases to saturate the fixed number of application servers, then all users receive a poor experience.

In order to address these problems, applications are adopting component-based "Service Oriented Architecture" (SOA) design patterns, whereby the application is decomposed into multiple tiers, each running on either physical or virtual servers. A typical deployment pattern would be a client desktop using a modern browser, talking to a web-server tier (e.g. running Apache), which talks to a business logic tier (e.g. running Java), which talks to a database tier (e.g. running IBM DB2). Each of these tiers can then be configured to run as a resource pool, by leveraging load balancers, J2EE servers, or parallel databases and database replication. Added benefits come from deploying these tiers using Virtual Machines instead of on the physical servers, as this provides greater flexibility and agility.

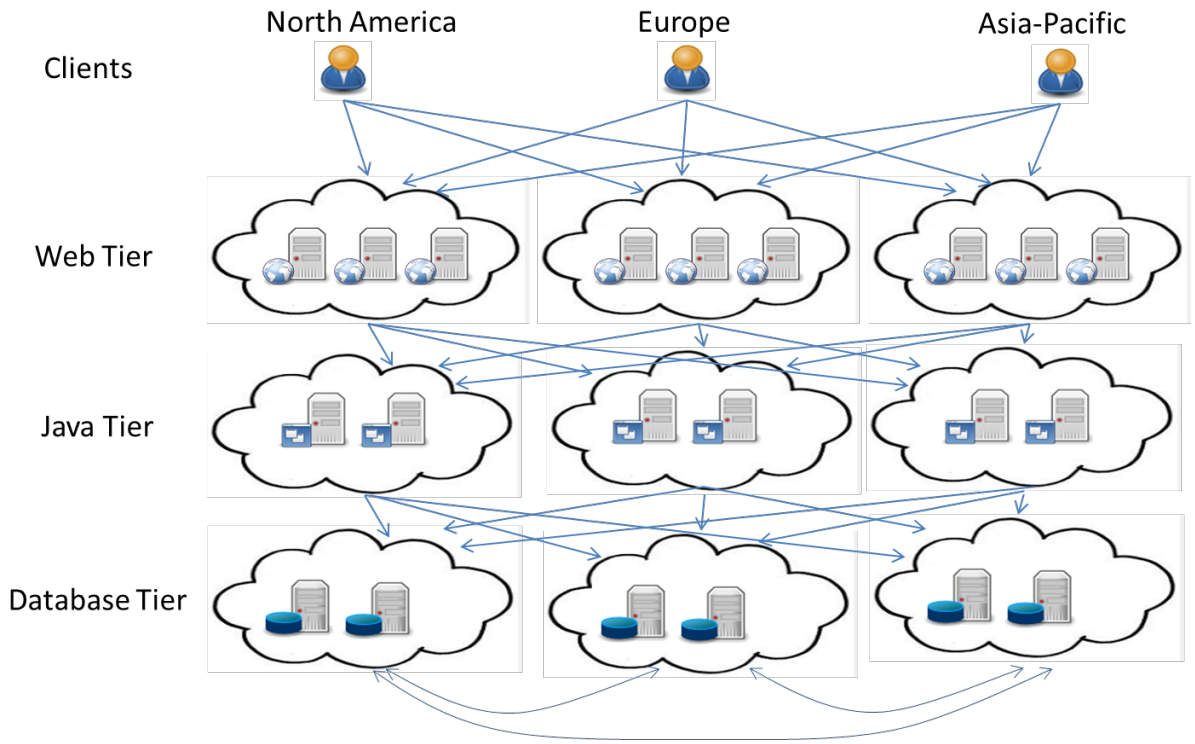
## The Challenge

For this challenge, we want to introduce you to a fictitious trading application called Hermes (named after the Greek god of boundaries, travel, communication and trade). This fictitious application runs in North America (NA), Europe (EU), and Asia-Pacific (AP), and is built on a three-tier SOA architecture. Hermes handles transactions from clients on a 24x7 non-stop basis. The application has a Web layer, a Java layer and a Database layer, comprised of virtual machines running in the three trading regions.

Your challenge is to write an algorithm to control how many virtual machines should be running within Hermes at any moment of the day, in what SOA tiers, and in which locations. To showcase your algorithm, you will compete in a trading simulation game, collecting as many trades as possible in order to maximize revenue.

Don't worry! We will explain everything you need to know to be successful in this challenge.

## The Hermes Application




This is a schematic diagram of the Hermes application. Clients access Hermes from North America (NA), Europe (EU), and Asia-Pacific (AP) at any time of the day or night. They access a web front-end hosted on one of several web servers in the Web Tier. As the application is quite interactive, the latency between the client and the Web Server is important. Wherever there is spare capacity, the web load balancers will distribute the client load locally (i.e. AP clients to AP web servers, and same for EU and NA). If the local web servers approach capacity, the load balancers will automatically redirect the traffic over onto an out-of-region web server (if there is spare remote capacity). As this happens, the application will fail to capture some transactions (due to clients not preferring the slow response from Hermes and either not trading or going to a competitor). If the client traffic cannot be redirected, then it will be dumped onto the local web server. When this happens, the local web servers will become extremely slow, and many transactions will be lost. This is a bad thing! As the web servers run independently, you should think about running web servers in each location able to handle most of the traffic locally.

For the transactions that are successfully handled on the Web Tier, they will get distributed to the Java Tier for processing. The locality of the Java server to the web server is somewhat less important than the client to the web, but it's still important to align the demand if possible. Like the web servers, the Java servers also run independently, so you can distribute these to the three locations to match the local demand.

In the course of processing the transaction, the Java server will talk to the Database Tier to access trading data (e.g. prices, stock symbols, etc.), and to place the trade. The database tier is less sensitive to location than the Java servers. Within a single site, you can run multiple instances of the database server as needed to handle the load. It's a parallel database, so it scales approximately linearly with load. The database needs to synchronize its state across all instances, so whilst this is treated as a negligible

performance penalty when the instances are in a single site, you will see a performance penalty running the database stretched across two sites, and a much larger penalty if you try to stretch it across all three sites. You should think carefully about where you want the database to run (given the current load conditions), and when moving it, start instances on the new site and then shut down on the old site, so as to minimize this cross-site database performance penalty.

## The Global VM Placement Engine

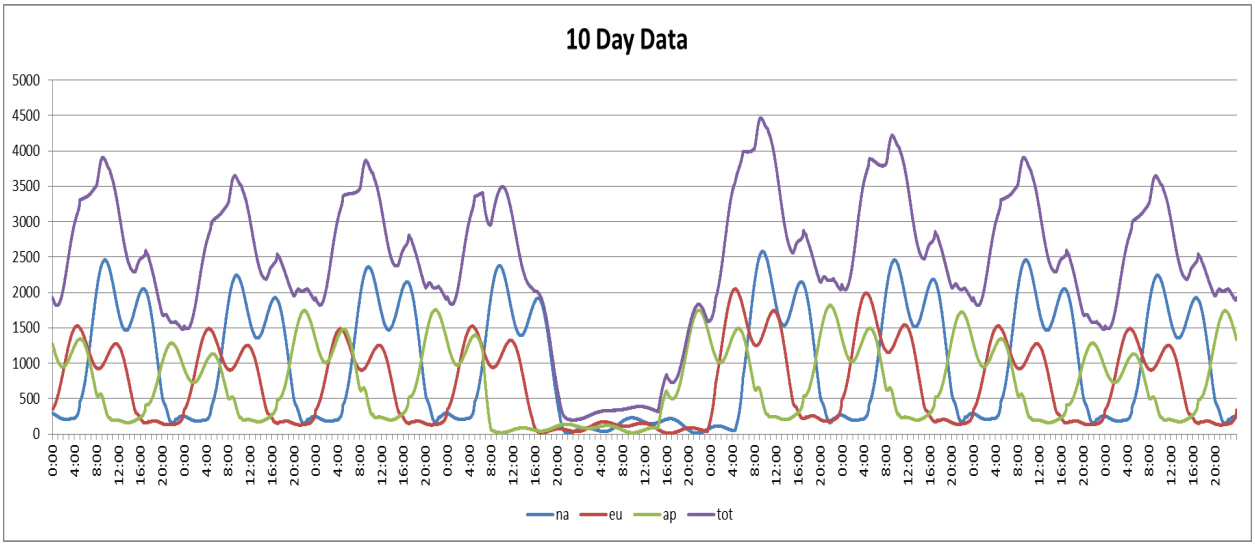
You will be writing an application comprised of two major components:

a) A basic client, with which you connect with the game server and perform basic interactions with the game server
b) A control algorithm by which you make dynamic choices about the infrastructure layout, placement, and distribution of the virtual machines (VMs) that comprise the Hermes application

The game server will run on a central PC, laptop or server. It communicates to you through the REST interface. It reads game data from a control file, executes the game control algorithm, and puts the resulting scores into a file for later judging.

The game is a turn-based simulation. Each turn represents 30 seconds of trading activity. On each turn, you will receive the current configuration (i.e. how many web, java and database servers are running in each site), the client transaction demand for the previous period (split into three demand numbers, one per site), the distribution data (showing how the demand was distributed to each tier and location), and the profit or loss for that period. Your job is then to send back your desired control inputs, stopping or starting a number of web servers in a location of your choice. The game simulates between 6 hours and 2 weeks of trading activity, always in 30-second intervals.

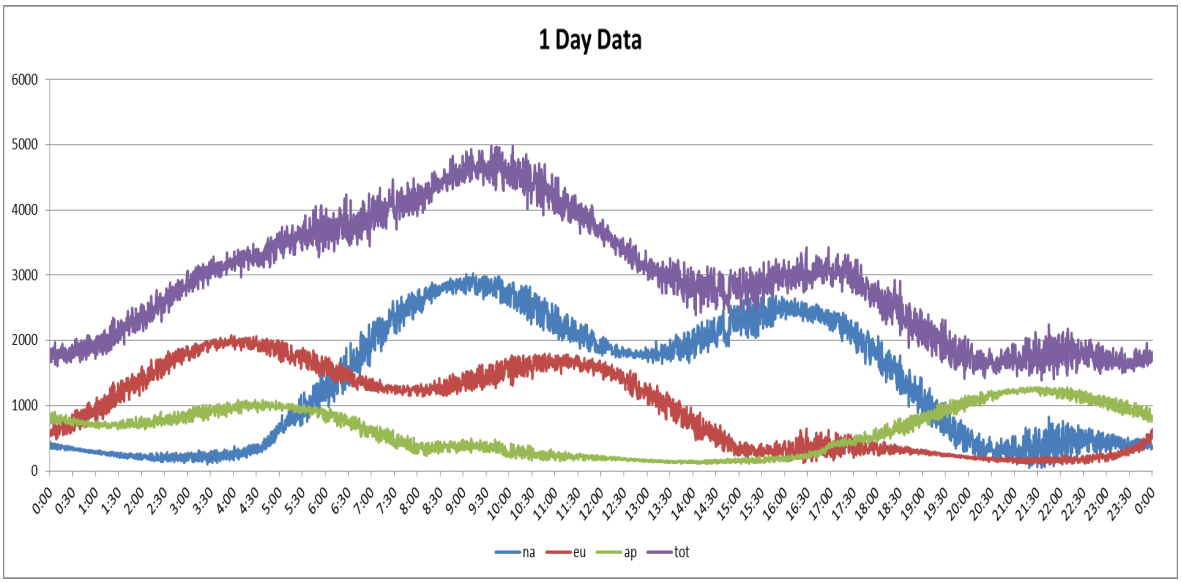
Here is a sample of the underlying trading pattern:




You will notice that each day has a morning peak at around 9am, and an afternoon peak at around 4pm. The EU peaks occur 5 hours earlier than the NA peaks (due to time zone), and are slightly smaller than the NA peaks. The AP peaks happen 7 hours before the EU peaks, and are usually slightly lower than the

EU peaks. You can also see what happens at the weekend, with AP dropping close to zero first, then EU, then NA. Note that for any specific day, the volumes can be higher or lower depending on market conditions.

You may assume this underlying pattern for your control algorithm (with daily differences in peak sizes per region). However, at a more granular level, there are peaks and troughs all the time, as shown here:




## Judging Criterion

1) Successful run of your application against the game server, completing the network conversation as documented herein (**20% weight)**
2) The amount of your final trading profit from a completed run of the game (**50% weight)**
3) The style and clarity of your source code (**10% weight)**
4) Your presentation, if applicable (**10% weight)**
5) The X Factor – anything unique and innovative about your solution (**10% weight)**

## Minimum Requirements

1) Your application can be written in a language of your choice – e.g. C++, Java, Python – and run on an OS and machine of your choice (e.g. PC, VM, Amazon EC2). The only requirement is that it will need network connectivity to the game server (located at http://hermes.wha.la)
2) You are free to include any GPL or LGPL open-source code into your application as you wish, so long as you fully credit the work to the owner, and you make clear what is yours and what is copied.
3) Your code has to be labeled as GPL or LGPL open-source.

# Game Details

## Playing the Game

As mentioned previously, you will be connecting to a game server that accepts a connection from a client that you will be creating, and then interacts with the game server on a turn-by-turn basis, which plays a game file provided by the Morgan Stanley team. The game file will be read line-by-line until all turns represented in the game file have been relayed to the user. Each line has information regarding how many possible "capture-able" transactions had arrived at each tier of each region, and based on the sophistication of your algorithm, your client will have captured either all or a portion of those trade requests that came into the previous turn as a direct result of your algorithm's adjustments to the infrastructure (which anticipated the needed level of infrastructure demand).

The game server will be hosted on an AWS (Amazon Web Service) instance for the weekend of the challenge. Each "game turn" represents 1 remote procedure (or REST) call, which corresponds to 1 minute of a trading day and 1 line in the CSV game file.

**Time Frame**

| **Game Time** | **Game Turns** |
|---|---|
| 1 minute | 1 turn |
| 1 day | 1,440 turns |
| 1 week | 10,080 turns |
| 4 weeks | 40,320 turns |
| 16 weeks | 161,280 turns |

Communication between the client and server will be in JSON. The game server will respond to the client after each game turn with a JSON string describing the state of the game. Information in this JSON will include, but is not limited to, the number of trades that were successfully captured by Hermes, number of servers running in each region for each tier, the previous turn's trade demand, and the cost per server (adjusted for research options and infrastructure upgrades, which will be explain in more detail in later sections of this document).

The client will then respond with a JSON string after each game turn with the control matrix (how many servers to bring up, to shut down, or keep the same for each tier in each region), the research decisions (which topic of research to start investing in), and the infrastructure upgrade decisions (which components to upgrade in the infrastructure). At the end of each turn, the team's net profit/loss is calculated by taking (profit constant) *(successful # of transactions) – the cost of the number of servers one has running.

Just to reiterate: the ultimate goal of the game is for your team to efficiently balance performance with infrastructure costs by dynamically controlling how many servers you have in any region for any tier. The infrastructure upgrades and research options change the dynamic of the global environment by either making servers cheaper or increasing capacity for each server for the rest of the game.

## Server Capacity Limit Behavior

The student will need to adjust the number of Web, Java and Database servers running to accommodate the change in workload demand for each region. The following tables represent the load thresholds of each infrastructure tier.

By default[1], each Web server will have the following capacity limits:

| Web Server Performance Table | | |
|---|---|---|
| Transactions | Successful | Comments: |
| 0-180 | 100% | Nominal web performance. All successful. |
| 181-200 | 90% | Web server "at capacity": 10% of transactions being lost. |
| 201-250 | 50% | Web server overloaded: 50% loss. |
| 251+ | 0% | Web server hung: 100% loss. |

By default[1], each Java server will have the following capacity limits:

| Java Server Performance Table | | |
|---|---|---|
| Transactions | Successful | Comments: |
| 0-400 | 100% | Nominal Java performance. All successful. |
| 401-500 | 90% | Java server "at capacity": 10% of transactions being lost. |
| 501-600 | 50% | Java server overloaded: 50% loss. |
| 601+ | 0% | Java server hung: 100% loss. |

By default[1], each Database server will have the following capacity limits:

| Database Server Performance Table | | |
|---|---|---|
| Transactions | Successful | Comments: |
| 0-1000 | 100% | Nominal database performance. All successful. |
| 1001-1200 | 90% | Database server "at capacity": 10% of transactions being lost. |
| 1201-1400 | 50% | Database server overloaded: 50% loss. |
| 1401+ | 0% | Database server hung: 100% loss. |

[1]These values are "default" because the player will have the opportunity to invest in research and upgrades during the course of the game that may alter some or all of these values.

## Transaction Failover Behavior

Once a server goes beyond 50% transaction loss (take the webserver for example - beyond 250, by default), before it fully hangs, the extra transactions will fail-over to its nearest region. If both alternative regions can't handle the load either, then that is when the transactions will return to its local server and hang. The following tables represent the distribution of transactions failing over to other regions for each tier.

By default[1], the Web tier will have the following transaction loss when failing over to another region:

| Client Distribution Performance Table | | |
|---|---|---|
| Client-to-Web | Successful | Comments: |
| Same Region | 100% | Nominal web responsiveness. (App feels like client expects.) |
| EU to/from NA | 70% | 30% loss for Trans-Atlantic web serving. (App feels slow.) |
| AP to/from NA | 50% | 50% loss for Trans-Pacific web serving. (App feels really slow.) |
| AP to/from EU | 50% | 50% loss for Trans-Asia web serving. (App feels really slow.) |

By default[1], the Java tier will have the following transaction loss when failing over to another region:

| Web Distribution Performance Table | | |
|---|---|---|
| Web-to-Java | Successful | Comments: |
| Same Region | 100% | Nominal Java responsiveness. |
| EU to/from NA | 90% | 10% loss. |
| AP to/from NA | 80% | 20% loss. |
| AP to/from EU | 80% | 20% loss. |

By default[1], the Database tier will have the following transaction loss when failing over to another region:

| Java Distribution Performance Table | | |
|---|---|---|
| Java-to-DB | Successful | Comments: |
| Same Region | 100% | Nominal Database responsiveness. |
| EU to/from NA | 90% | 10% loss. |
| AP to/from NA | 90% | 10% loss. |
| AP to/from EU | 90% | 10% loss. |

[1]These values are "default" because the player will have the opportunity to invest in research and upgrades during the course of the game that may alter some or all of these values.

## Database Replication Behavior

In addition to regional failovers, the game also simulates the concept of database replication (used in mitigating regional latency in real-world implementations). If you have databases in more than one region, those databases have to sync with each other in order to stay consistent. As soon as you run the database across two regions, the synchronization and transaction-commit overhead starts to hurt performance. In the worst case, where the database is concurrently active in three sites, you will suffer a compounding effect of trying to three-phase commit the transactions globally.

| Database Split-Site Performance Table | | |
|---|---|---|
| DB loss | Successful | Comments: |
| Same Region | 100% | No losses for running inside one region. |
| EU to/from NA | 80% | 20% loss due to DB synching trans-Atlantic. |
| AP to/from NA | 70% | 30% loss due to DB synching trans-Pacific. |
| AP to/from EU | 70% | 30% loss due to DB synching trans-Pacific. |
| Triple-site | 39% | = 80% * 70% * 70% success rate. Worst overhead possible! |

## Starting/Shutting Down Servers

When the control algorithm makes a change to the infrastructure, any change will take a certain amount of time before the change goes into full effect:

| Tier | Start Time | Stop Time | Comments |
|---|---|---|---|
| Web | 2 turn | 0 turns | Two game turns to start, immediate stop. |
| Java | 4 turns | 1 turn | Slower start than web, one turn to stop. |
| Database | 8 turns | 2 turns | Slower start and stop than Java tier. |

**Notes:**

1) **You are paying for the cost of the server when it is starting up, when it runs and when it is shutting down.**

2) **There is a hard-limit of 300 servers/tier, for each region, that you can have in your infrastructure.**

3) **There is a hard-limit of 50 servers/tier, for each region, that can that can be added or subtracted in any one turn.**

## Infrastructure Upgrades

The students will have the option of investing their profits towards upgrading their infrastructure (assuming they are making enough profit to afford the upgrade). Each type of upgrade affects the global infrastructure in different ways. Costs go into effect as soon as the control algorithm says to start the upgrade, even if the upgrade isn't fully implemented yet. The following are the different types of upgrades that the student can pursue:

| Component | Default Configuration |
|---|---|
| **Network** | 1 Gbps |
| **RAM** | 1GB |
| **CPU** | 1 Core |

| Component | Level 1 Upgrade | Server Cost after Level 1 Upgrade | Effect on Infrastructure |
|---|---|---|---|
| **Network** | 10Gbps | 20% cost increase per server | 33% increase in capacity/server |
| **RAM** | 2GB | | |
| **CPU** | 2 Cores | | |

| Component | Level 2 Upgrade | Server Cost after Level 2 Upgrade | Effect on Infrastructure |
|---|---|---|---|
| **Network** | 100 Gbps | 16% cost increase per server | 33% increase in capacity/server |
| **RAM** | 4GB | | |
| **CPU** | 4 Cores | | |

## Research Upgrades

The students will have the option of investing their profits towards research (assuming they are making enough profit to afford it). Each research topic affects the global infrastructure in different ways. **During the research phase, the cost of doing research will be $5,000/turn and will take 10 days to complete (10,400 turns).** Please make these values easily configurable in your implementation.

The following are the different types of research areas that the student can pursue:

| | Effect on Infrastructure | Example |
|---|---|---|
| **Grid** | Lowers server cost by ~55%/server | If your servers cost $1,000 each, with **Grid** they would cost $450 each |
| **Green** | Lowers server cost by ~30%/server | If your servers cost $1,000 each, with **Green** they would cost $700 each |
| | Increases transaction capacity/server by ~30% | If your server capacity is 100 each, with **Green** your capacity is 130 |
| **Low Latency** | Increases transaction capacity/server by ~55% | If your server capacity is 100 each, with **Low Latency** your capacity is 155 |
| **WAN Compression** | Decreases 50% of Transactions lost when failing across regions | If you currently lose 50% when failing over from NA - AP, with **WAN Compression** you only lose 25% |
| **DB Replication** | Decreases 50% of Transactions lost when synchronizing databases | If you currently lose 30% when failing over from NA - AP, with **DB Replication** you only lose 15% |

**Notes:**

1) **At the beginning of the game, you won't have enough money to invest in research or upgrades. So the team has to first make enough profit to be able to spend the money towards research and upgrades.**

2) **If during research, a team goes into the negative in terms of net revenue, the research phase is terminated and all research progress is lost. The team would have to start again from the beginning if they want to invest in research.**

# API Documentation

## Hermes Gameplay API

### Description

This operation is used to send commands to the Hermes game server.

- **Base URL:** http://hermes.wha.la/api/hermes
- **HTTP Type**: POST

### Input Type

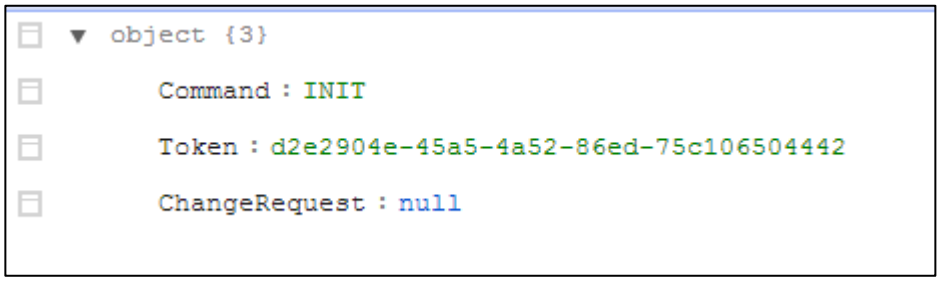
Input to the operation is a JSON object with following properties:

| Property Name | Description | Data Type |
|---|---|---|
| Command | The command which needs to be sent to the server. See ***Command Types*** for various possible commands. | String |
| Token | Unique token code provided to the team by the administrator. A valid token code is to be passed with each request. | String |
| ChangeRequest | This JSON object is used to change the nodes in the server tiers across the regions. This command is also used to upgrade the infrastructure levels and research upgrades. | ChangeRequest Object |

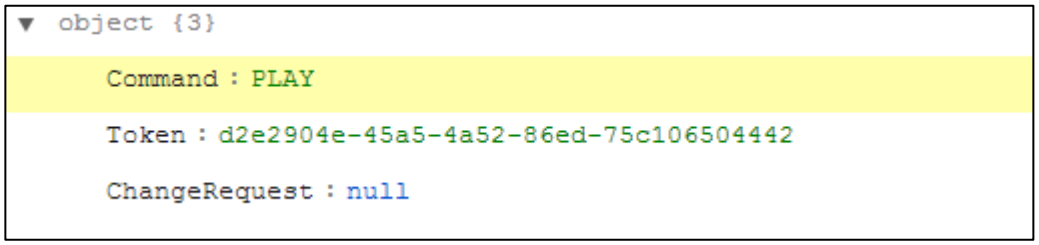
### Sample Request Objects

The following images show the input object with Command, token and a ChangeRequest object.

**INIT Command Sample:**




**PLAY Command Sample:**



**CHNG Command Sample:**

```
Command : CHNG
Token : 00000000-0000-0000-0000-000000000000
▼ ChangeRequest {3}
  ▼ Servers {3}
    ▼ WEB  {1}
      ▼ ServerRegions {2}
        ▼ EU   {1}
            NodeCount : -2
        ▼ NA   {1}
            NodeCount : -2
    ▼ JAVA       {1}
      ▼ ServerRegions {1}
        ▼ NA   {1}
            NodeCount : 2
    ▼ DB   {1}
      ▼ ServerRegions {1}
        ▼ AP   {1}
            NodeCount : -2
    UpgradeInfraStructure : true
    UpgradeToResearch : GRID
```

In the sample request above, following operation is being requested:

1) CHNG command is issued with the token.
2) 2 server nodes are removed from WEB tier in EU region.
3) 2 server nodes are removed from WEB tier in NA region.
4) 2 server nodes are added from JAVA tier in NA region.
5) 2 server nodes are removed from DB tier in AP region.
6) Infrastructure is upgraded to the next level.
7) Research will be upgraded to the GRID level.

### Output Type

Each call to the operation will return a JSON object with following properties:

| Property Name | Description | Data Type |
|---|---|---|
| RequestStatus | This variable will provide the status of last executed command. Possible values returned from server are:<br>SUCCESS: Last request was successful.<br>FAILURE: Last request was un-successful with an unknown error.<br>VALIDATE_FAILURE: Last request was un-successful. More details on the error will be provided in ErrorMessage property described below. | String |
| ErrorMessage | This object will provide the error details in case the request status is FAILURE. | GameError Object |
| ServerState | This object returns the state of the server after the effects of the last request. | GameServerFarmState Object |

## Command Types

The various types of values which are allowed as command type are as follows:

1. **INIT**: "INIT" as command is used to initialise a token. A successful response from server on execution of this command means that token is authorised at server and is ready to be sent to server with all subsequent commands.

   Value of the Command input parameter should be "INIT".
   A valid **token** is a **mandatory** input parameter to the POST operation with above command.
   **ChangeRequest** input parameter is **optional** to the POST operation.

2. **PLAY:** "PLAY" as a command will play the next turn on the server with the nodes configuration present on the game server for the specified team.

   Value of the Command input parameter should be "PLAY".
   A valid **token** is a **mandatory** input parameter to the POST operation with above command.
   **ChangeRequest** input parameter is **optional** to the POST operation.

3. **CHNG:** "CHNG" as a command is used to change the nodes configuration on the game server. It is also used to upgrade infrastructure and perform research.

Value of the Command input parameter should be “CHNG”.
A valid **token** is a **mandatory** input parameter to the POST operation with above command.
**ChangeRequest** input parameter is **mandatory** to the POST operation.

## ChangeRequest Object

This object is used to specify the addition/ deletion of server nodes from layers across regions. The structure of the object is described below:

| Property Name | Description | Data Type |
|---|---|---|
| Servers | This is a named array of ***InfrastructureUpgradeLevel Object*** and Tier name. For all tier names like WEB, JAVA and DB, there will be ***InfrastructureUpgradeLevel Object*** for each of them in the Servers array.<br><br>Sample:<br>In the sample below, Servers is an array of WEB, JAVA, and DB server tier objects<br>"Servers": {<br>"WEB": {<br>"ServerRegions": {<br>"EU": {<br>"NodeCount": -2<br>}<br>}<br>}, "JAVA": {<br>"ServerRegions": {<br>"NA": {<br>"NodeCount": -2<br>}<br>}<br>}, "DB": {<br>"ServerRegions": {<br>"AP": {<br>"NodeCount": -2<br>}<br>}<br>}<br>} | array |

| Property Name | Description | Data Type |
|---|---|---|
| UpgradeInfraStructure | This flag if set to true will upgrade the infrastructure level to the next level. Setting this flag as true will cost the amount needed for the upgrade to next level. The various infrasturcture levels available are "DEFAULT", "LEVEL1", and "LEVEL2". The default infrastructure level is "DEFAULT". The downgrade of infrastructure levels is not supported. | bool |
| UpgradeToResearch | This variable is used to upgrade the Research to the next level. The various research levels available are "DEFAULT", "GRID", "GREEN", "LOW_LATENCY","WAN_COMPRESSION" and "DB_REPLICATION". Default configuration has UpgradeToResearch = "DEFAULT".<br><br>Example: If the research level is to be upgraded from DEFAULT to GRID, then the value of the variable UpgradeToResearch should be set as "GRID". | String |

## GameError Object

This object is provides the error details in case the Request status is "FAILURE".

| Property Name | Description | Data Type |
|---|---|---|
| ErrorNo | This variable returns the error number. | String |
| ErrorMessage | This variable returns the error description. | String |

## Error Codes & Messages

| Error Code | Error Text |
|---|---|
| ERR 101 | `Server State Not Found For Client Id: {0}` |
| ERR 102 | `Client Not Found Client Id: {0}` |
| ERR 103 | `Config Not Initiliazed` |
| ERR 104 | `Play Not Found For Client Id: {0}, Turn No: {1}` |
| ERR 105 | `Infrastructure upgrade is already in progress {0}. Change` |

| Error Code | Error Text |
|---|---|
| | operation failed. |
| ERR 106 | All available infrastructure upgrades consumed. Change operation failed. |
| ERR 107 | Research Upgrade has already been applied {0}. Change operation failed. |
| ERR 108 | All available infrastructure upgrades consumed. Change operation failed. |
| ERR 109 | Research Upgrade is already in progress {0}. Change operation failed. |
| ERR 110 | Incorrect Research Upgrade request!! |
| ERR 111 | TOKEN MISSING. |
| ERR 112 | INVALID COMMAND. |
| ERR 113 | Unknown Error Occurred. Error Message :{0} |
| ERR 114 | Client Not Active Client Id: {0} |
| ERR 115 | Insufficient profit accumulated for Research upgrade. Change operation failed. |
| ERR 116 | Insufficient profit accumulated for Research. Current research cancelled. |
| ERR 117 | Synch Mode On. Await next turn. Current turn no: {0} |
| ERR 118 | JSON Input Validation Error: {0} |
| ERR 119 | Max Node count reached error on applying the change in tier region: {0} |

## GameServerFarmState Object

This object is provides the current state of the Server. It has the latest data including the after effects of the last successful transaction. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| TransactionTime | This variable returns transaction time of the last | Date |

| Property Name | Description | Data Type |
|---|---|---|
| | successful request. | |
| TurnNo | This variable returns turn number which was successfully executed in the last request. | Int |
| CostPerServer | This variable cost to add a new server node. | Int |
| ProfitConstant | Profit per successful transaction. | Int |
| ProfitAccumulated | This variable returns profit that was accumulated in the till successful request. | Int |
| ProfitEarned | This variable returns profit that was earned in the last successful request. | Int |
| ServerTiers | This is a named array of ***InfrastructureUpgradeLevel Object*** and Tier name. For all tier names like WEB, JAVA, and DB, there will be ***InfrastructureUpgradeLevel Object*** for each of them in the Servers array. | array |
| InfraStructureUpgradeLevels | This is a named array of ***InfrastructureUpgradeLevel Object*** for Infrastructure upgrades | array |
| ResearchUpgradeLevels | This is a named array ***ResearchUpgradeLevel Object*** for Infrastructure upgrades | array |
| InfraStructureUpgradeState | Represents the state of the infrastructure upgrade | array |
| ResearchUpgradeState | Represents the state of the research upgrade | array |

## ResearchUpgradeLevel Object

This object is used to specify the research upgrade level. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| Name | Represents the name of the research | string |
| UpgradeCost | The cost involved in performing the research | Int |
| No.OfTurnsRequired | The no. of turns to be completed to perform the research | Int |

| Property Name | Description | Data Type |
|---|---|---|
| CostPerServerBenefit | Cost savings on per server basis | Int |
| TransactionBenefit | Transaction benefit with the research upgrade | Int |
| FailOverBenefit | Fail over benefit with the research upgrade | Int |
| DBReplicationBenefit | Database replication benefit with the research upgrade | Int |

## InfrastructureUpgradeLevel Object

This object is used to specify the infrastructure upgrade level.Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| Name | Represents the name of the infrastructure | string |
| UpgradeCost | The code involved in upgrading the infrastructure | Int |
| No.OfTurnsRequired | The no. of turns to be completed to upgrade the infrastructure | Int |
| TransactionBenefit | Transaction benefit after the upgrade | int |

## ServerRegion Object

This object is used to specify the server configuration for a specific server region and tier. This object is specific for a region like EU or NA or AP for a specific tier. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| NodeCount | This variable is used to get or set the number of server nodes. Positive number in request object will increase the node counts and negative number will decrease the node counts. | int |
| NoOfTransactionsInput | This variable returns the number of transactions that were played on the last turn. This is an output only field and will be found in the response. | int |
| NoOfTransactionsExecuted | This variable returns the number of | int |

| Property Name | Description | Data Type |
|---|---|---|
| | transactions that were executed on the last turn on this tier in this region. This is an output only field and will be found in the response. | |
| NoOfTransactionsSuceeded | This variable returns the number of transactions that were successfully executed on the last turn on this tier in this region. This is an output only field and will be found in the response. | int |
| NodeChangeState | This is an array of ***NodeChangeState*** Object . This is an output only field and will be found in the response. | NodeChangeState |

## ServerPerformance Object

This object is used to specify the tier name and its corresponding nodes configurations across various regions. This object is specific for a tier like WEB or JAVA or DB for a specific tier. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| CapactityLevels | This is an array of ***Capacity Object*** | Array |
| FailOverLevels | This is an array of ***FailOver Object*** | Array |
| DBReplicationLevels | This is an array of ***FailOver Object*** | Array |

## NodeChangeState Object

This object is used to specify the tier name and its corresponding nodes configurations across various regions. This object is specific for a tier like WEB or JAVA or DB for a specific tier. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| NodeCount | Can be positive or negative node change count. | Int |

| Property Name | Description | Data Type |
|---|---|---|
| EffectiveAfterTurns | Node change effective after configured no. of turns. | Int |

## Capacity Object

This object specifies the capacity of the server. This object is specific for a tier like WEB or JAVA or DB for a specific tier. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| UpperLimit | Max transactions supported by the server | Int |
| LowerLimit | Min transactions supported by the server | Int |
| SuccessPercentage | Percentage of transactions successfully processed | Int |
| IsAtOverLoad | Returns if the server is overloaded | bool |

## FailOver Object

This object specifies the FailOver of the server. This object is specific for a tier like WEB or JAVA or DB for a specific tier. Fields are described below:

| Property Name | Description | Data Type |
|---|---|---|
| RegionTo | Region to which failover will happen | string |
| RegionFrom | From Region which the failover will happen | string |
| SuccessPercentage | Percentage of transactions successfully processed | Int |

## Game Server Assumptions

A few assumptions & additional rules for the game logic:

### *1. Infrastructure Upgrades*

Are sequential – Default -> Level 1 -> Level 2

### *2. Research Upgrades*

1) Research upgrades are parallel. A research upgrade can be followed by another research upgrade, even before the benefits of the earlier upgrade are implemented.
2) User cannot request for the same research upgrade twice.
3) Research upgrade costs will be considered for profit calculation till that turn when the benefits are applied.

### 3. *Increase in Server Capacity*

When a research upgrade is made and the server capacity is enhanced as in the case of Low Latency, the increase in capability, is as calculated as given below:

*Initial Capacity levels:*

| Number of Transactions | Success Percentage |
|---|---|
| 1-180 | 100% |
| 181-200 | 90% |
| 201-250 | 50% |
| 251- | 0% |

*And say, after a benefit of 100 transactions per server capacity,*

| Number of Transactions | Success Percentage |
|---|---|
| 1-280 | 100% |
| 281-300 | 90% |
| 301-380 | 50% |
| 381 | 0% |

## 4. *Failover benefits*

When a research upgrade is made to WAN Compression, the failover benefits are calculated as given below:

*Initial Failover Percentages:*

| Region | Success Percentage |
|---|---|
| EU to/from NA | 70% |
| AP to/from NA | 50% |
| AP to/from EU | 50% |

After say a 50% failover benefit

| Region | Success Percentage |
|---|---|
| EU to/from NA | 85% |
| AP to/from NA | 75% |
| AP to/from EU | 75% |

## 5. *DB Replication*

If, for example, the DB replication percentages are:

- EU - NA - 80%
- NA - AP - 70%
- EU - AP - 70%

1) If there are servers in EU region and NA region, 80% is the success rate
2) If there are servers in NA and AP or EU and AP success rate is 70%
3) If we have servers in all three regions, success rate is 80*70*70%

## 6. *Failover Order*

The first failover is always to the nearest region, where the loss due to failover is the lowest:

1) EU will first failover to NA then to AP
2) NA first to EU then to AP
3) AP first to NA then to EU