EZapp (Owners: Ben, Beshoi)
EZapp is the entrypoint for input data from outside of the EZstack ecosystem. The majority of data communication from within EZapp to other EZstack services leverages Kafka streams to create a producer/consumer ecosystem.

The data transferred between EZapp, the Query Aggregator, and the denormalizer consist of two general data types. This type of data gets placed on different Kafka topics so raw JSON data with minimal metadata info is all that is needed to be submitted to the different input streams.

Type 1: Input Data
Example 1:

```
{
        “userId” : 123,
        “firstName” : “bob”
        “familyId” : 25
}
```

Example 2:

```
{
        “familyId” : 25,
        “address” : “123 Main St”
}
```

Type 2: Query
Example 1:

```
{
…
“Query” : {
                “tableName” : “name”,
                “Filter” : (...),
                “Join” : {
                        “Query” : (...),
                        “Replace” :  [
                                {
                                        “outerTableField” : “attributeName”,
                                        “innerTableField” : “attributeName”
                                }]
                }
        }
}
```

Example 2:

```
{
…
“Query” : {
        “Field” : “fieldName”,
        “Value” : “some value”,
        “Op” : “eq”,
        “Condition” : {
                “conditionOp” : “AND”,
                “Query” : {...}
                }
        }
}
```

Logging (Owners: Beshoi)
The underlying logging system is a REST interface.

POST /logger/insert
Example 1:
INPUT:

```
{
        “type” : info”,
        “message” : “info message regarding your system”
}
```

OUTPUT:

```
{
        “result” : “success”
}
```

Example 2:
INPUT:

```
{
        “type” : “warning”,
        “message” : “warning message regarding your system”
}
```

OUTPUT:

```
{
        “result” : “success”
}
```

Example 3:

INPUT:
{
	“type” : “error”,
	“message” : “error message regarding your system”
}

OUTPUT:
{
	“result” : “success”
}

POST /logger/bulk
Example:
INPUT:
[{
	“type” : info”,
	“message” : “info message regarding your system”
},
{
	“type” : info”,
	“message” : “info message regarding your system”
},
{
	“type” : info”,
	“message” : “info message regarding your system”
}]

OUTPUT:
{
	“result” : “success”
}

GET /logger

Attribute	Definition
type	The type of info that should be returned. Can be left blank to return all types.
last	Specifies the amount of records to be returned.

Example:
/logger?type=info&last=5

OUTPUT:
[{
	“type” : info”,
	“message” : “info message regarding your system”
},
{
	“type” : info”,
	“message” : “info message regarding your system”
} …]

Samza Mesos (Owners: Beshoi)
The following application is utilized by the development team to run Samza jobs. The denormalizer and Query aggregation components both leverage the following interface to start their Samza Jobs.

The configuration files developed need to follow the format specified in the samza mesos repo on git which is located here:
https://github.com/gw-cs-sd/sd-18-ezstack/tree/samza-mesos/samza-mesos

The jobs can be run and status can be retrieved using Samza StreamJob interface located:
http://samza.apache.org/learn/documentation/0.13/jobs/job-runner.html

All the rules are cached on Zookeeper by EZapp in the same JSON format submitted by the Aggregator. The syntax for the JSON rules is provided here again for convenience.

```
{
	“joinedTable” : [“table1”, “table2”, “table3”, ...],
	“removeAttributes” : [
		{
			“table” : “table1”,
			“attributes” : [“attribute1”, “attribute2”, ...]
		}
		{
			“table” : “table2”,
			“attributes” : [“attribute1”, “attribute2”, ...]
		}
		...
	]
}
```

Denormalizer (Owners: Ben)
The denormalizer also leverages Kafka by consuming data provided from EZapp as well as producing data to be stored in elasticsearch and other services. The denormalizer produces two major types of data. As with all other products that leverage Kafka in our system, the data gets separated into different Kafka topics depending on type.

Type 1: Unmodified Input Data + EZstack Metadata
Example 1:

```
{
        “table” : “tableName”,
        “key” : “Unique Key”,
        “timestamp” : “uuid”,
        “data” {
                “userId” : 123,
                “firstName” : “bob”
                “familyId” : 25
        }
        “version” : 1
}
```

Example 2:

```
{
        “table” : “tableName”,
        “key” : “Unique Key”,
        “timestamp” : “uuid”,
        “data” {
                “familyId” : 25,
                “address” : “123 Main St”
        }
        “version” : 1
}
```

Type 2: Denormalized Data
Example:

```
{
        “table” : “denormalizedTableName”,
        “key” : “Unique Key”,
        “timestamp” : “uuid”,
        “data” {
                “userId” : 123,
                “firstName” : “bob”,
                “family” :  {
```

```
                        “familyId” : 25,
                        “address” : “123 Main St”
                }
        }
        “version” : 1
}
```

Query Aggregation (Owners: Sam)

The Aggregator also consumes the same input Kafka streams that the Denormalizer consumes from EZapp for its analytics and rule generation algorithm. Unlike the rest of our system that produces data via Kafka streams and forgets it, the Aggregator posts new rules to EZapp via an internal REST interface once it has reached a conclusion.

POST /aggregator/rule
Example:
INPUT:

```
{
        “joinedTable” : [“table1”, “table2”, “table3”, ...],
        “removeAttributes” : [
                {
                        “table” : “table1”,
                        “attributes” : [“attribute1”, “attribute2”, ...]
                }
                {
                        “table” : “table2”,
                        “attributes” : [“attribute1”, “attribute2”, ...]
                }
                ...
        ]
}
```

GET /aggregator/rule
Returns all the rules generated realistically even in a large scale environment the rules shouldn’t exceed a few hundred since that might add more overhead on data denormalization then there are benefits.