Crazylvan Documentation

Release 1.0.0

AO

Jul 21, 2018

Contents:

Go Home

CHAPTER 1

Docker

The easiest way to get started with CrazyIvan is with Docker

If you do not have Docker installed, please visit the link above to get setup before continuing.

The first thing we need to do is setup the Docker Network that will allow us to communicate between our containers:

```
docker network create dvs
```

Before we can start CrazyIvan, we need to have a few other programs running first. Luckily, these can all be setup with Docker as well:

```
docker run -d --name=registry --network=dvs consul
```

```
docker run -d --publish=7474:7474 --publish=7687:7687 --env=NEO4J_AUTH=none
--volume=$HOME/neo4j/data:/data --network=dvs --name=database neo4j
```

```
docker run -i -t -d -p 2181:2181 -p 9092:9092 --env ADVERTISED_PORT=9092 --env
ADVERTISED_HOST=queue --name=queue --network=dvs spotify/kafka
```

This will start up a single instance each of Neo4j, Kafka, and Consul. Consul stores our configuration values, so we'll need to set those up. You can either view the Consul Documentation for information on starting the container with a Web UI, or you can use the commands below for a quick-and-dirty setup:

```
docker exec -t registry curl -X PUT -d 'neo4j://graph-db:7687' http://
localhost:8500/v1/kv/ivan/DB_ConnectionString
```

```
docker exec -t registry curl -X PUT -d 'queue:9092' http://localhost:8500/v1/
kv/ivan/KafkaBrokerAddress
```

```
docker exec -t registry curl -X PUT -d 'True' http://localhost:8500/v1/kv/
ivan/StampTransactionId
```

```
docker exec -t registry curl -X PUT -d 'Json' http://localhost:8500/v1/kv/
ivan/Data_Format_Type
```

Then, we can start up CrazyIvan:

```
docker run --name crazyivan --network=dvs -p 5555:5555 -d aostreetart/
crazyivan -consul-addr=registry:8500 -ip=localhost -port=5555
-log-conf=CrazyIvan/log4cpp.properties
```

This will start an instance of CrazyIvan with the following properties:

- Connected to network 'dvs', which lets us refer to the other containers in the network by name when connecting.
- Listening on localhost port 5555
- Connected to Consul Container

We can open up a terminal within the container by:

```
docker exec -i -t crazyivan /bin/bash
```

The 'stop_crazyivan.py' script is provided as an easy way to stop CrazyIvan running as a service. This can be executed with:

```
python stop_crazyivan.py hostname port
```

For a more detailed discussion on the deployment of CrazyIvan, please see the *Deployment Section* of the documentation.

CHAPTER 2

Using the Latest Release

In order to use the latest release, you will still need to start up the applications used by CrazyIvan, namely Neo4j, Kafka, and Consul. This can be done using the docker instructions above, or by installing each to the system manually. Instructions: * Neo4j * Consul

Then, download the latest release from the Releases Page

Currently, pre-built binaries are available for:

- Ubuntu 16.04
- CentOS7

Unzip/untar the release file and enter into the directory. Then, we will use the easy_install.sh script to install CrazyIvan. Running the below will attempt to install the dependencies, and then install the CrazyIvan executable:

```
sudo ./easy_install.sh -d
```

If you'd rather not automatically install dependencies, and only install the executable, then you can simply leave off the '-d' flag. Additionally, you may supply a '-r' flag to uninstall CrazyIvan:

```
sudo ./easy_install -r
```

Once the script is finished installing CrazyIvan, you can start CrazyIvan with:

```
sudo systemctl start crazyivan.service
```

The 'stop_crazyivan.py' script is provided as an easy way to stop CrazyIvan running as a service. This can be executed with:

```
python stop_crazyivan.py hostname port
```

Note: The CrazyIvan configuration files can be found at /etc/crazyivan, and the log files can be found at /var/log/crazyivan.

CHAPTER 3

Building from Source

The recommended system for development of CrazyIvan is either Ubuntu 16.04 or CentOS7. You will need gcc 5.0 or greater installed to successfully compile the program.

```
git clone https://github.com/AO-StreetArt/CrazyIvan.git
```

```
mkdir crazyivan_deps
```

```
cp CrazyIvan/scripts/linux/deb/build_deps.sh crazyivan_deps/build_deps.sh
```

```
cd crazyivan_deps
```

```
./build_deps.sh
```

You will be asked once for your sudo password.

```
cd ../CrazyIvan
```

```
make
```

This will result in creation of the crazy_ivan executable, which we can run with the below command:

```
./crazy_ivan
```

When not supplied with any command line parameters, CrazyIvan will look for an ivan.properties file and log4cpp.properties file to start from.

You may also build the test modules with:

```
make tests
```

In order to run CrazyIvan from a properties file, you will need:

- You will also need to have a Neo4j Server installed locally. Instructions can be found at https://neo4j.com/developer/get-started/

Continue on to the *Configuration Section* for more details on the configuration options available when starting CrazyIvan.

CHAPTER 4

Configuration

4.1 Properties File

Crazy Ivan can be configured via a properties file, which has a few command line options:

- `./crazy_ivan` - This will start Crazy Ivan with the default properties file, ivan.properties
- `./crazy_ivan -config-file=file.properties` - This will start Crazy Ivan with the properties file, file.properties. Can be combined with -log-conf.
- `./crazy_ivan -log-conf=logging.properties` - This will start Crazy Ivan with the logging properties file, logging.properties. Can be combined with -config-file.

The properties file can be edited in any text editor.

4.2 Consul

Crazy Ivan can also be configured via a Consul Connection, in which we must specify the address of the consul agent, and the ip & port of the Inbound ZeroMQ Connection.

- `./crazy_ivan -consul-addr=localhost:8500 -ip=localhost -port=5555` - Start Crazy Ivan, register as a service with consul, and configure based on configuration values in Consul, and bind to an internal 0MQ port on localhost
- `./crazy_ivan -consul-addr=localhost:8500 -ip=tcp://my.ip -port=5555 -log-conf=logging.properties` - Start Crazy Ivan, register as a service with consul, and configure based on configuration values in Consul. Bind to an external 0MQ port on tcp://my.ip, and configure from the logging configuration file, logging.properties.

We can also use both a properties file and a Consul connection, in which case the properties file is used to define the ip and port of the inbound ZeroMQ connection, while Consul is used for registration and all other configuration retrieval.

- `./crazy_ivan -consul-addr=localhost:8500 -config-file=file.properties`

When configuring from Consul the keys of the properties file are equal to the expected keys in Consul.

4.3 Logging

The Logging Configuration File can also be edited with a text file, and the documentation for this can be found [here] (http://log4cpp.sourceforge.net/api/classlog4cpp_1_1PropertyConfigurator.html). Note that logging configuration is not yet in Consul, and always exists in a properties file.

Two logging configuration files are provided, one for logging to the console and to a file (log4cpp.properties), and another to log to syslog and to a file (log4cpp_syslog.properties). Both show all of the logging modules utilized by Crazy Ivan during all phases of execution, and all of these should be configured with the same names (for example, log4cpp.category.main).

Crazy Ivan is built with many different logging modules, so that configuration values can change the log level for any given module, the log file of any given module, or shift any given module to a different appender or pattern entirely. These modules should always be present within configuration files, but can be configured to suit the particular deployment needs.

4.4 Startup

Crazy Ivan can be started with an option to wait for a specified number of seconds prior to looking for configuration values and opening up for requests. This is particularly useful when used with orchestration providers, in order to ensure that other components are properly started (in particular, in order to allow time for Consul to be populated with default configuration values).

- `./crazy_ivan -wait=5` - This will start Crazy Ivan with the default properties file, and wait 5 seconds before starting.

4.5 Configuration Key-Value Variables

Below you can find a summary of the options in the Properties File or Consul Key-Value Store:

4.5.1 DB

- DB_ConnectionString - The string used to connect to the Neo4j instance (example: neo4j://neo4j:neo4j@localhost:7687)

4.5.2 0MQ

- 0MQ_InboundConnectionString - The connectivity string for the inbound 0MQ Port

4.5.3 Kafka Connection

- KafkaBrokerAddress - The address of the Kafka connection to monitor

4.5.4 Behavior

- StampTransactionId - True to stamp Transaction ID's on messages, False if not. Transaction ID's are passed on Inbound Responses and Outbound messages, in order to link the two together.
- Data_Format_Type - JSON to accept JSON messages, protobuf to accept protocol buffer messages

Go Home

CHAPTER 5

API Overview

The CrazyIvan API utilizes either JSON or Protocol Buffers, based on what the server is configured to process. In either case, the field names and message structure remains the same. This document will focus on the JSON API, but with this knowledge and the DVS Interface Protocol Buffer files, the use of the Protocol Buffer API should be equally clear.

Response Messages follow the same format as inbound messages.

To start with, here is an example JSON message which will create a single scene, and register a user device to it:


```
{
    “msg_type”:4,
    “transaction_id”:”123465”,
    “scenes”:[
    {
    “key”:”jklmnop”,
    “name”:”Test Scene 10”,
    “latitude”:124.0,
    “longitude”:122.0,
    “distance”:100.0,
    “region”:”Test Region”,
    “assets”:[“Test Asset 1”, “Test Asset 2”],
    “tags”:[“Test Tag 1”, “Test Tag 2”],
    “devices”:[
    {
    “key”:”Ud_132”,
    “transform”:{
```

"translation":[1.0,1.0,1.0],

"rotation":[1.0,1.0,1.0]

}

}

]

}

]

}

Let's take a look at the individual fields.

5.1 Scene List

The Scene List is the highest level wrapper in the API. It only contains 5 keys, one of which is an array of scenes.

- msg_type – 0 for create scene, 1 for update scene, 2 for retrieve/query scene(s), 3 for delete scene, 4 for device registration, 5 for device de-registration, and 6 for device alignment. The message type applies to all objects in the objects array.
- transaction_id – An ID to distinguish a transaction within a larger network of applications
- scenes – An array containing scenes
- err_code – Integer error code, full list of values can be found below in the appendix
- err_msg – A string error message, containing a human-readable description of the issue

5.2 Scene

A single Scene , is represented by a single element of the array from the "scenes" key of the scene list.

- key – Scene Key value (UUID)
- name – Name of the Scene
- latitude – A float value representing the latitude of the Scene. Used for distance-based queries.
- longitude – A float value representing the longitude of the Scene. Used for distance-based queries.
- distance – A float value that is only required for distance based queries. With this, we can query Crazy Ivan for scenes within a specific distance of a lat/long position.
- num_records – An Integer value which represents the maximum number of scenes that can be returned from a query to Crazy Ivan
- devices – ID For the Scene containing the object
- region – The Region containing the Scene
- tags – String Tags which can be used to query for Scenes
- assets – ID's for assets used for the Scene

5.3 User Device

A single device is represented by a single element of the array from the “devices” key of the scene.

- key – Device Key value (UUID)
- hostname - The hostname of the device, for use in UDP communications
- port - The port of the device, for use in UDP communications
- connection_string - An optional additional connectivity string for UDP Communications
- transform – A transformation object which represents the transformation from the scene coordinate system to the device coordinate system.

5.4 Transformation

A transformation is represented by the object in the “transform” key of the device.

- translation – An array of 3 floats representing x, y, and z values for a translation
- rotation – An array of 3 floats representing x, y, and z values for a local euler rotation

CHAPTER 6

Field Mapping

Field	Type	Create	Get	Update	Delete	Register	Leave	Align
msg_type	Integer	X	X	X	X	X	X	X
err_code	Integer							
err_msg	String							
transaction_id	String	*	*	*	*	*	*	*
num_records	String		*					
key (scene)	String		*	X	X	X	X	X
name	String	X	*	*		*		
latitude	Float	X	*	*		*		
longitude	Float	X	*	*		*		
distance	Float		*	*				
region	String	*	*	*		*		
assets	Array - String	*	*	*		*		
tags	Array - String	*	*	*		*		
key (device)	String					X	X	X
connection_string	String					*		
hostname	String					*		
port	Integer					*		
translation	Array - Double					*	*	*
rotation	Array - Double					*	*	*

X – Required

* - Optional

CHAPTER 7

Message Types

7.1 Scene Create

Create a new Scene. Returns a unique key for the scene.

7.2 Scene Retrieve

The scene retrieve message will retrieve a scene by key, and return the full scene. It can also be used to run queries against other scene attributes, as well as perform distance-based queries to find scenes within a certain radius of a given lat/long coordinate.

7.3 Scene Update

Scene updates can be used to update scene attributes.

7.4 Scene Destroy

Destroy an existing Scene by key. Basic success/failure response.

7.5 Device Register

Register a device to a scene. If no transformation is supplied, then CrazyIvan will respond with an initial guess on what the correct transform is.

7.6 Device De-Register

De-Register a device to a scene.

7.7 Device Align

Apply a correction to the transformation currently stored between a scene and user device.

7.8 Device Retrieve

Retrieve the connectivity information of a user device.

CHAPTER 8

Appendix A: JSON Message Samples

8.1 Inbound

8.1.1 Scene Create

{ “msg_type”:0, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[

{ “key”:”jklmnop”, “name”:”Test Scene 10”, “latitude”:124.0, “longitude”:122.0, “distance”:100.0, “region”:”TestRegion5”, “assets”:[“TestAsset10”], “tags”:[“Testing2”]

}

]

}

8.1.2 Scene Retrieve

{ “msg_type”:2, “transaction_id”:”123464”, “scenes”:[

{ “key”:”ijklmno”

}

]

}

8.1.3 Scene Update

{ “msg_type”:1, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[

{ “key”:”jklmnop”, “name”:”Test Scene 101”, “latitude”:126.0, “longitude”:129.0, “distance”:110.0, “region”:”TestRegion20”, “assets”:[“TestAsset20”], “tags”:[“Testing4”]

```
            }
        ]
}
```


8.1.4 Scene Destroy


```
{
    “msg_type”:3,
    “transaction_id”:”123464”,
    “scenes”:[
    {
    “key”:”ijklmno”
    }
    ]
}
```


8.1.5 Device Registration


```
{ “msg_type”:4, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[
        { “key”:”jklmnop”, “name”:”Test Scene 10”, “latitude”:124.0, “longitude”:122.0, “distance”:100.0,
            “devices”:[
                { “key”:”Ud_132”, “transform”:{
                    “translation”:[1.0,1.0,1.0], “rotation”:[1.0,1.0,1.0]
                }
                }
            ]
        }
    ]
}
```


8.1.6 Device De-Registration


```
{ “msg_type”:5, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[
        { “key”:”jklmnop”, “name”:”Test Scene 10”, “latitude”:124.0, “longitude”:122.0, “distance”:100.0,
            “devices”:[
                { “key”:”Ud_132”, “transform”:{
                    “translation”:[1.0,1.0,1.0], “rotation”:[1.0,1.0,1.0]
                }
                }
```

```
                ]
            }
        ]
}
```


8.1.7 Device Alignment


```
{ “msg_type”:6, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[
            { “key”:”jklmnop”, “name”:”Test Scene 10”, “latitude”:124.0, “longitude”:122.0, “distance”:100.0,
                “devices”:[
                    { “key”:”Ud_132”, “transform”:{
                        “translation”:[6.0,1.0,1.0], “rotation”:[1.0,45.0,1.0]
                      }
                    }
                ]
            }
        ]
}
```


8.1.8 Device Retrieval


```
{ “msg_type”:7, “err_code”:100, “err_msg”:”Test”, “transaction_id”:”123465”, “scenes”:[
            {
                “devices”:[
                    { “key”:”Ud_132”
                    }
                ]
            }
        ]
}
```


8.2 Response

8.2.1 Scene Create


```
{ “msg_type”:0, “err_code”:100, “num_records”:1, “scenes”:[
            { “key”:”ijklmno”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “assets”:[], “tags”:[], “devices”:[]
            }
        ]
```

}

8.2.2 Scene Retrieve

{ “msg_type”:2, “err_code”:100, “transaction_id”:”123465”, “num_records”:1, “scenes”:[
{ “key”:”jklmnop”, “name”:”Test Scene 10”, “region”:”TestRegion5”, “latitude”:124.0, “longitude”:122.0, “distance”:0.0, “assets”:[], “tags”:[“Testing2”], “devices”:[]
}
]
}

8.2.3 Scene Update

{ “msg_type”:1, “err_code”:100, “num_records”:1, “scenes”:[
{ “key”:”ijklmno”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “asset_ids”:[], “tags”:[], “devices”:[]
}
]
}

8.2.4 Scene Destroy

{ “msg_type”:3, “err_code”:100, “num_records”:1, “scenes”:[
{ “key”:”hijklmn”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “asset_ids”:[], “tags”:[], “devices”:[]
}
]
}

8.2.5 Device Registration

{ “msg_type”:4, “err_code”:100, “transaction_id”:”123465”, “num_records”:1, “scenes”:[
{ “key”:”jklmnop”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “asset_ids”:[], “tags”:[], “devices”:[
{ “key”:”Ud_132”, “transform”:{“translation”:[0.0,0.0,0.0],”rotation”:[0.0,0.0,0.0]}
}
]
}
]
}

8.2.6 Device De-Registration

{ “msg_type”:5, “err_code”:100, “transaction_id”:”123464”, “num_records”:1, “scenes”:[
{ “key”:”ijklmno”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “asset_ids”:[], “tags”:[], “devices”:[]
}
]
}

8.2.7 Device Alignment

{ “msg_type”:6, “err_code”:100, “transaction_id”:”123465”, “num_records”:1, “scenes”:[
{ “key”:”jklmnop”, “latitude”:0.0, “longitude”:0.0, “distance”:0.0, “asset_ids”:[], “tags”:[], “devices”:[]
}
]
}

8.2.8 Device Retrieval

CHAPTER 9

Appendix B: Error Codes

const int NO_ERROR = 100

const int ERROR = 101

const int NOT_FOUND = 102

const int TRANSLATION_ERROR = 110

const int PROCESSING_ERROR = 120

const int BAD_MSG_TYPE_ERROR = 121

const int INSUFF_DATA_ERROR = 122

Go Home

CHAPTER 10

Deployment

Note: At this time, CrazyIvan has no built-in security or encryption mechanisms. Until such time, it is not recommended to deploy CrazyIvan in Production.

The easiest methodology of deployment for CrazyIvan is using Docker. At this time, it has not been tested with either Docker Compose or Docker Swarm.

This page will be updated after larger scale testing has been performed with CrazyIvan.

Go Home

CHAPTER 11

Architecture

This is designed to be used as a microservice within a larger architecture. This will take in CRUD messages for scenes, as well as track user device registrations.

A .proto file is included to allow generating the bindings for any language (the [protocol buffer compiler] (https://developers.google.com/protocol-buffers/) is installed by the build_deps script), which can be used to communicate via protocol buffers.

Please note that running Crazy Ivan requires an instance of both Neo4j and Kafka to connect to in order to run.

Crazy Ivan can also be deployed with Consul as a Service Discovery and Distributed Configuration architecture. This requires the Consul Agent to be deployed that Crazy Ivan can connect to.

CHAPTER 12

Design

In order to allow for real-time, distributed visualization, one of the key problems that needs to be solved is ensuring that coordinate systems between various user devices and objects are synchronized. A key abstraction in this case is a ‘Scene’ which is an arbitrary collection of objects and devices. A device can register/de-register from any scene, as well as apply corrections to the coordinate system relationship between it and the scene it’s registered to.

This is done by storing relationships between scenes and devices, and then using these to build relationships between scenes themselves. When devices move between these scenes, they will apply corrections. As they apply corrections, we will build a set of known mappings between scenes which should allow users to move without needing any corrections by returning the pre-calculated differences.

CHAPTER 13

Object Change Streams

Object Change Streams ensure that all registered User Devices remain up to date about objects within their scenes. Crazy Ivan monitors a Kafka Topic, which is populated by CLyman upon receipt of Object Updates. Crazy Ivan picks up these messages and sends them out to the registered devices via UDP.

Go Home

CHAPTER 14

Dependencies

Go Home

CrazyIvan is built on top of the work of many others, and here you will find information on all of the libraries and components that CrazyIvan uses to be successful.

Licenses for all dependencies can be found in the licenses folder within the repository.

14.1 CppKafka

CppKafka is a wrapper on top of librdkafka, which provides quick and easy access to pushing Kafka messages.

CppKafka is released under a BSD License.

14.2 ZeroMQ

Zero MQ is a lightweight messaging library that CrazyIvan uses to communicate. It is fast, versatile, and has bindings for many major languages.

Zero MQ is released under an LGPL License.

14.3 CppZmq

CppZmq is the C++ binding for libzmq, which was written in C.

CppZmq is released under an MIT License.

14.4 Log4cpp

Log4Cpp is a logging library based on Log4j.

Log4Cpp is released under an LGPL License.

14.5 Eigen

Eigen is a Linear Algebra library.

Eigen is released under an MPL License.

14.6 RapidJson

RapidJson is a very fast JSON parsing/writing library.

RapidJson is released under an MIT License.

14.7 AO Shared Service Library

AOSSL is a collection of C++ wrappers on many of the C libraries listed here.

AOSSL is released under an MIT License.

14.8 LibHiredis

LibHiredis is used to communicate with Redis, a distributed key-value store, and is a dependency of AOSSL

LibHiredis is released under a BSD License.

14.9 LibNeo4j

LibNeo4j is used to communicate with Neo4j, a Graph Based Database.

LibNeo4j is released under an Apache 2 License.

14.10 LibUUID

LibUUID is a linux utility for generating Universally Unique ID's.

LibUUID is released under a BSD License.

14.11 LibCurl

LibCurl is a ubiquitous networking library.

LibCurl is released under an MIT License.

14.12 LibProtobuf

LibProtobuf and the Protocol Buffer Compiler comprise a serialization system which CrazyIvan can use to communicate in lieu of JSON. You can find more information about Protocol Buffers at the Google Developer Site

The Protocol Buffer License is unique yet very unrestrictive. For more information please see the license itself

14.13 DVS Interface

Finally, we also depend on the DVS Interface Library which houses a collection of .proto files for this project.

DVS Interface is released under an MIT License.

14.14 Automatic Dependency Resolution

For Ubuntu 16.04 & Debian 7, the build_deps.sh script should allow for automatic resolution of dependencies.

14.15 Other Acknowledgements

Here we will try to list authors of other public domain code that has been used:

```
René Nyffenegger - Base64 Decoding Methods
```

CHAPTER 15

Developer Notes

This page contains a series of notes intended to be beneficial for any contributors to Crazy Ivan.

15.1 Development Docker Image

Generating a development Docker Image is made easy by the DebugDockerfile. This image is unique in that it does not enter directly into Crazy Ivan, but rather installs all of the necessary dependencies and then waits.

First, execute the below command from the root folder of the project to build your local debug image: `docker build --no-cache --file DebugDockerfile -t "aostreetart/crazyivan:debug" .`

Once this completes, run your image with the below command: `docker run --name crazyivan -p 5555:5555 -d aostreetart/crazyivan:debug`

You can update the port number to whatever you like, and keep in mind that you may also need to connect the container to a docker network, depending on your configuration. For example: `docker run --name crazyivan --network=dvs -p 5555:5555 -d aostreetart/crazyivan:debug`

Finally, you can open up a terminal within the box with: `docker exec -i -t crazyivan /bin/bash`

The container will have Crazy Ivan and all it's dependencies pre-installed, so you can get right to work!

15.2 Generating Releases

The release_gen.sh script is utilized to generate releases for various systems. It accepts three command line arguments:
* the name of the release: crazyivan-*os_name-os_version* * the version of the release: we follow semantic versioning
* the location of the dependency script: current valid paths are linux/deb (uses apt-get) and linux/rhel (uses yum)

Read About Crazy Ivan Automated Testing

Go Home

CHAPTER 16

Automated Testing

Crazy Ivan uses Travis CI for automated testing.

Within the Travis CI Configuration, several steps are executed to complete full functional testing:

- Set up Docker instances of Neo4j and Consul, and then populate the KV Store in Consul with several configuration values.
- Build a new Docker Image for Crazy Ivan and start it.
- Download 0-Meter. This is a custom tool developed for 0MQ load testing, and is used to send a series of messages to Crazy Ivan over the course of the tests. The configuration for 0-Meter CI Tests can be found in the ci/ folder.
- Run 0-Meter to send a series of messages, some expected to fail and others to succeed, to Crazy Ivan. Validate the err_code field in the response.
- If all tests pass, then push the newly built image to Docker Hub.

Note that unit tests are performed within the Dockerfile itself, so that the Docker build will fail if any unit tests fail. If you are adding unit tests to Crazy Ivan, you should add them within the Dockerfile as well.

Go Home

CHAPTER 17

Crazy Ivan

17.1 Overview

Crazy Ivan is a service designed to store 'scenes', which means an arbitrary collection of objects in 3-space within a geographic area. Devices can register/de-register from scenes as they move through the world, and as they do we build a network of relationships that can be used to determine the transformations needed for other devices.

Crazy Ivan also serves as a UDP Server, communicating Object Change Streams to registered devices.

Detailed documentation can be found on ReadTheDocs.

17.2 Features

- Storage of Scenes (Groups of virtual objects & user devices associated to a latitude/longitude)
- Efficient calculation of coordinate system transformations based on existing data
- Means to store manual corrections from users
- Connect to other services over Zero MQ using Google Protocol Buffers.
- Scalable microservice design

Crazy Ivan is a part of the AO Aesel Project, along with CLyman. It therefore utilizes the DVS Interface library, also available on github. It utilizes the Scene.proto file for inbound communications.

Stuck and need help? Have general questions about the application? Reach out to the development team at crazyivan@emaillist.io