Persistence Plugin Architecture (Proposal v0.0.1)

Revision History

Date	Who	What
20190102	Andre	Initial draft
20190103	Trevor	Added clarifications
20190106	Andre	v0.0.1
20190117	Trevor	Added Discussion Notes

** Updated Discussion Notes **

On 17-Jan-2019 we discussed the following proposal in the Core Working Group call. Full recording and meeting minutes can be found here. The current plan of action for Edinburgh is as follows:

- We will defer usage of plugins until a later development cycle, Fuji at the earliest.
- For Edinburgh, we are committed to supporting Mongo and Redis at the persistence layer for relevant services within the edgex-go repo. These implementations will be a part of the edgex-go source tree, as they were for the Delhi release.
- Andre at Redis has committed to closing any necessary gaps to support all relevant services with a tentative delivery of 15-Mar-2019
- An alternate approach to plugins was proposed by ObjectBox (see #corewg-persist channel on Slack) which would utilize Go modules to externalize 3rd party implementations. Ivan / Markus at ObjectBox agreed to provide a POC for the group demonstrating this approach. Once that is ready, we will cover it in a subsequent Core Working Group call.
- Resources at Microsoft and Intel are conferring with regard to the open issue detailing the lack of plugin support for Windows. Hoping for traction here leading to acceptance for a future Go version.

Summary

The Persistence Plugin Architecture is intended to address decoupling persistence implementations from the EdgeX Foundry code base. The core-data, core-metadata, export-client, support-logging and support-notifications services all currently rely on database persistence. Today (Delhi release), persistence implementations are part and parcel of the EdgeX Golang Services repo. As a result, adding a new persistence implementation or updating an existing implementation requires all team members to have knowledge of every supported persistence platform as well as access to an installation of each in order to fully test any changes. New capabilities must be added to every persistence implementation before they can be called complete. This is not scalable. For a future (Edinburgh) release, the goal is to move persistence implementations into their own repos so they can be maintained independently of the EdgeX code base.

The proposed architectural solution utilizes Go Plugins.

Within each service, a configuration value will drive which persistence plugin to load, similar to what we do now. This value can be the same across all services or even different for every service should a customer feel that one or another platform is best suited to the role the service plays in their environment. Each service defines an interface for the necessary database operations (core-data, core-metadata) and then a vendor's plugin must conform to and support that interface.

User Stories

1. As a persistence implementation provider (*a provider*), I want a mechanism so that I can add a new persistence implementation to the EdgeX platform.
2. As a provider, I want a mechanism for sustaining a persistence implementation so that I do not rely on EdgeX platform releases.
3. As a provider, I want a mechanism for configuring my implementation without knowing the rest of the EdgeX platform configuration.
4. As a provider, I want the EdgeX platform build scripts to consolidate my build-time and run-time requirements so that I can maintain my implementation independently of the EdgeX platform.
5. As a provider, I want an API contract so that I know what to implement.
6. As a provider, I want an API contract so that I can unit test.
7. As a provider, I want EdgeX platform testing to be able to certify implementation so that EdgeX platform users will have the confidence to use my implementation.
8. As an EdgeX platform maintainer (*a maintainer*), I want a default persistence implementation so that I can maintain CI/CT.
9. As a maintainer, I want providers to create reasonable defaults so that I can add them into CI/CT.
10. As an EdgeX platform user (*a user*), I want to select one or more persistence implementations for use by the various EdgeX services.
11. As a user, I want the option to run the platform along with the selected persistence implementation(s) directly on the OS.
12. As a user, I want the option to run the platform along with the selected persistence implementation(s) in a containerized deployment.

Design

The design has two roles: the provider role and the platform role.

Provider

The provider maintains their plugin in their own repository along with the build scripts to create the plugin, the EdgeX build fragments needed by the platform build, and the EdgeX Compose file fragments needed to create the EdgeX Compose file.

Repo Organization

The provider repo follows the Go standard project layout. In addition, providers are encouraged to maintain and distribute prebuilt binaries of their plugin.

Path	Description
`pkg/NAME.so.VERSION`	The plugin lives in the pkg directory where *NAME* is the plugin name and VERSION follows Semantic Versioning 2.0.
`pkg/NAME-docker-compose-fragment.VERSION.yml`	Addition to the Docker Compose file for the persistence service (aka the DB). Optional.
`pkg/NAME-config.VERSION`	Persistence service configuration file. Optional.

Note *NAME* and *VERSION* must match across plugin, compose file fragment, and configuration file.

Platform

The platform role is further divided into config time and run time. During config time, the configuration of the platform drives the consolidation of the Docker Compose file, possibly downloading the plugin, and possibly downloading the persistence configuration file. It is then assumed at run time, the compose file has mounted the volume where the persistence configuration file lives and the plugin is in a path known by the startup scripts.

Config Time

At config time, the plugin, the Docker Compose file fragment, and the persistence configuration file may need to be downloaded from the repo. This is obtained from the service `.toml` file or by querying Consul.

EdgeX Configuration

```
[Databases]
  [Databases.Primary]
  Host = 'localhost'
  Name = 'coredata'
  Username = 'scott'
  Password = 'tiger'
  Port = 1234
  Timeout = 5000
  Type = 'NAME.so'
  Version = '1.0.0'
  PluginSource = 'https://github.com/some-provider/edgex-plugin/blob/master/pkg'
```

In addition to overloading the **Type** key, two additional keys are added. The plugin, compose file fragment, and configuration file are copied to `cmd/SERVICE/plugins`.

Key	Description
Type	The Type key currently drives a switch statement in the service initialization code. If the key is not recognized in as a known value, it is assumed to name the plugin.
Version	The Version key is used to complete the name for the plugin as well as the name for the other dependencies.
PluginSource	The PluginSource key provides the path to the plugin as well the path for the other dependencies. It can be a URL or a local file path.

The compose file fragment is merged into the Docker Compose file for the service. If there is a configuration file it is mounted as part of the compose file.

pkg/docker-compose-fragment.yml

```
some-provider:
  image: some-provider:alpine
  ports:
    - "1234:1234"
  container_name: edgex-some-provider
  hostname: edgex-some-provider
  networks:
    - edgex-network
  volumes:
    - data:/data
  depends_on:
    - volume
```

Run Time

The assembled Docker Compose file will start the persistence service, if needed. When started, the EdgeX service attempts to use the **Databases.Primary.Type** key to reference a known builtin database. When the value is not recognized it will now attempt to load a plugin.

Acceptance Criteria

MVP

- Provider implementations in their own repo
- EdgeX platform config time scripts to create Docker Compose file
- Updated black box tests

v.Next

- Provider certification
- Provider included in platform CI/CT

Reference Patterns

The basis of the requirements came out of looking at how others have solved this challenge. Also, note Vladimir Vivien's helpful article on "Writing Modular Go Programs with Plugins".

Azure IoT Edge

Azure IoT Edge is organized around independently configured modules (aka containers) that are grouped to form a solution. Each module is described independently and then combined into a solution with a deployment manifest. For example, the RedisEdge module is described using the following `module.json`.

module.json

```
{
    "$schema-version": "0.0.1",
    "description": "RedisEdge",
    "image": {
        "repository": "$CONTAINER_REGISTRY_ADDRESS/redis-edge",
        "tag": {
            "version": "1.0.0",
            "platforms": {
                "amd64": "./Dockerfile.amd64",
                "arm32v7": "./Dockerfile.arm32v7",
                "arm64v8": "./Dockerfile.arm64v8"
            }
        },
        "buildOptions": []
    },
    "language": "javascript"
}
```

To build a solution with this module, a deployment manifest is constructed using scripts that parse a deployment template which enumerates the modules along with their configuration. Note the use environment variables; these are populated from a `.env` file that is not part of the source repository.

deployment.template.json

```
{
  "modulesContent": {
    "$edgeAgent": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "runtime": {
          "type": "docker",
          "settings": {
            "minDockerVersion": "v1.25",
            "loggingOptions": "",
            "registryCredentials": {
              "myiotregistery": {
                "username": "$CONTAINER_REGISTRY_USERNAME",
                "password": "$CONTAINER_REGISTRY_PASSWORD",
                "address": "$CONTAINER_REGISTRY_ADDRESS"
              }
            }
          }
        },
        "systemModules": {
          "edgeAgent": {
            "type": "docker",
            "settings": {
              "image": "mcr.microsoft.com/azureiotedge-agent:1.0",
              "createOptions": ""
            }
          },
          "edgeHub": {
            "type": "docker",
            "status": "running",
            "restartPolicy": "always",
            "settings": {
              "image": "mcr.microsoft.com/azureiotedge-hub:1.0",
              "createOptions": "{\"HostConfig\":{\"PortBindings\":{\"5671/tcp\":[{\"HostPort\":\"5671\"}], \"
8883/tcp\":[{\"HostPort\":\"8883\"}],\"443/tcp\":[{\"HostPort\":\"443\"}]}}}"
            }
          }
        },
        "modules": {
          "RedisEdge": {
            "version": "1.0",
            "type": "docker",
            "status": "running",
            "restartPolicy": "always",
            "settings": {
              "image": "${MODULES.RedisEdge.amd64}",
              "createOptions": "{\"HostConfig\": { \"PortBindings\": { \"6379/tcp\": [ { \"HostPort\": \"6379\"
} ] } } }"
            }
          }
        }
      }
    },
    "$edgeHub": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "routes": {},
        "storeAndForwardConfiguration": {
          "timeToLiveSecs": 7200
        }
      }
    }
  }
}
```

Which results in the following deployment manifest

deployment.json

```
{
  "modulesContent": {
    "$edgeAgent": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "runtime": {
          "type": "docker",
          "settings": {
            "minDockerVersion": "v1.25",
            "loggingOptions": ""
          }
        },
        "systemModules": {
          "edgeAgent": {
            "type": "docker",
            "settings": {
              "image": "mcr.microsoft.com/azureiotedge-agent:1.0",
              "createOptions": ""
            }
          },
          "edgeHub": {
            "type": "docker",
            "status": "running",
            "restartPolicy": "always",
            "settings": {
              "image": "mcr.microsoft.com/azureiotedge-hub:1.0",
              "createOptions": "{\"HostConfig\":{\"PortBindings\":{\"5671/tcp\":[{\"HostPort\":\"5671\"}], \"
8883/tcp\":[{\"HostPort\":\"8883\"}],\"443/tcp\":[{\"HostPort\":\"443\"}]}}}"
            }
          }
        },
        "modules": {
          "RedisEdge": {
            "version": "1.0",
            "type": "docker",
            "status": "running",
            "restartPolicy": "always",
            "settings": {
              "image": "andresandboxregistry.azurecr.io/redis-edge:1.0.0-amd64",
              "createOptions": "{\"HostConfig\": { \"PortBindings\": { \"6379/tcp\": [ { \"HostPort\": \"6379\"
} ] } } }"
            }
          }
        }
      }
    },
    "$edgeHub": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "routes": {},
        "storeAndForwardConfiguration": {
          "timeToLiveSecs": 7200
        }
      }
    }
  }
}
```