System Management high-level API (re-)design

Jim White/12/31/18

This document follows a lot from what is in https://github.com/edgexfoundry/edgex-go/pull/772/files. Where it deviates, we are trying not to provide for all the executor options (non-scalable) but allow these other executors to exist via the call to an independently created external executor in a standard/interface way – this per our 11/13/18 sys management WG call.

Going forward, our start/stop/restart functionality will only support calls to either

- docker compose
- or some outside, undefined executable app that adheres to a defined API (per WG decision of 11/13/18).

These are referred to as the "executors" of the start, stop, and restart functionality. The latter option (some app) allows use of scripts inside an executable, call to OS specific technology like snaps or sysd, etc. without having to build all these options into the SMA – allowing for a more scalable solution in EdgeX but still allowing use of all sorts of implementation technology outside of EdgeX. In future releases, if other options are deemed important to the community and require built-in functionality, they can be added under the currently defined interfaces and structures.

The user's choice to use either a "docker compose" or "app" executor will be established by configuration option (currently OperationType is the config option using either "docker" or "app" to specify).

In the case that "app" executor is the chosen implementation for start/stop/restart functionality, another configuration option (appLocation) will specify the location and executable to be called to start, stop or restart. *[Should we have separate config options for path and executable? Should we have separate config options for start, stop and restart executables – would we need different executables for each?]*

The execution of an "app" executable will be OS dependent under the covers as the SMA will make a call to execute via GoLang's exec.Command.

EdgeX SMA API Trace

A request (by REST call) of the SMA to start, stop or restart service(s) begins at the router.

```
In internal/system/agent/router.go
        b.HandleFunc("/operation", operationHandler).Methods(http.MethodPost)
```

The router passes the request to the operationHandler function where action, services and parameters are determined *(yes – we keep parameters for now and keep it simple without another struct around all of these; the params may allow us to deal with enable/disable option = see below)*

```
In router.go
        operationHandler( )
                // invoke the operation and check the result
                err = InvokeOperation(o.Action, o.Services, o.Parameters)
```

The invocation function (InvokeOperation) then makes the call to the specific start, stop or restart interface on the executor to invoke the specific operation (with the applicable service name and parameters list) for each of the services.

```
In internal/system/agent/services.go
  func InvokeOperation(action string, services []string, params []string) error {
    for each service
      if Stop
        executorClient.(interfaces.ServiceStopper)
      if Start
        executorClient.(interfaces.ServiceStarter)
      if Restart
        executorClient.(interfaces.ServiceRestarter)
```

The executor functions (ServiceStopper, ServiceStarter and ServiceRestarter) all defined by internal/system/agent/interfaces/services.go.

The executor (aka executor client) is established on bootstrapping of the SMA. Specifically, it is defined in init.go.

```
In init.go
func newExecutorClient(operationsType string) (interfaces.ExecutorClient, error) {

  switch operationsType {
  case "app":
    return &executor.ExecuteApp{}, nil
  case "docker":
    return &executor.ExecuteDocker{}, nil
  default:
    return nil, nil
  }
}
```

Docker Compose Executor

The DockerCompose executor implementation will be defined in internal/system/agent/executor/docker.go. *This will use the start per docker-compose work that Akram now has working using the Docker-in-Docker image.*

```
In docker.go
  StartService(services string, params []strings){
    //ignore params for now
    call to start via docker-compose
  StopService(services string, params []strings){
    //ignore params for now
    call to stop via docker-compose
  ReStartService(services string, params []strings){
    //ignore params for now
    call to restart via docker-compose
```

App Executor

The app executor implementation will be defined in internal/system/agent/executor app.go. *This implementation will be very similar to what Ian has expressed in custom.go in his "large PR" #772. Akram – we'll have to provide a simple app that uses OS kill and run commands to provide prototype.*

```
In app.go
  StartService(services string, params []strings){
    call to executable setup by configuration with service name and params
    For demonstration - provide small, separately downloadable Linux based OS executable
that does OS proc.kill() (per os.go)
  StopService(services string, params []strings){
    call to executable setup by configuration with service name and params
    For demonstration - provide small, separately downloadable Linux based OS executable
that does launch of service executable
  ReStartService(services string, params []strings){
    call to executable setup by configuration with service name and params
    For demonstration - provide small, separately downloadable Linux based OS executable
that does OS proc.kill() (per os.go) and then does launch of service executable
```

Executable Interface

When using an external application to start, stop and restart the services, the executable must support the following parameters:

executable-name service-name action-name parameters

The parameters list would be separated by a space if there was more than one.
The execution output shall be returned to the SMA and the output treated as string output and returned.

The work Ian did to implement a snap executor in snap.go will be accomplished in a Canonical executable that follows this guideline.