Nutanix Webhooks Listener and Plugin Framework Tutorial

### 

Contents

[Introduction](#3znysh7)

Assumptions

Listener Plugin Architecture

Listener Plugin Workflow

Listener Library and Plugin Implementation Steps and Pseudo Code

Deploy and Run

Sample Log File

# Introduction

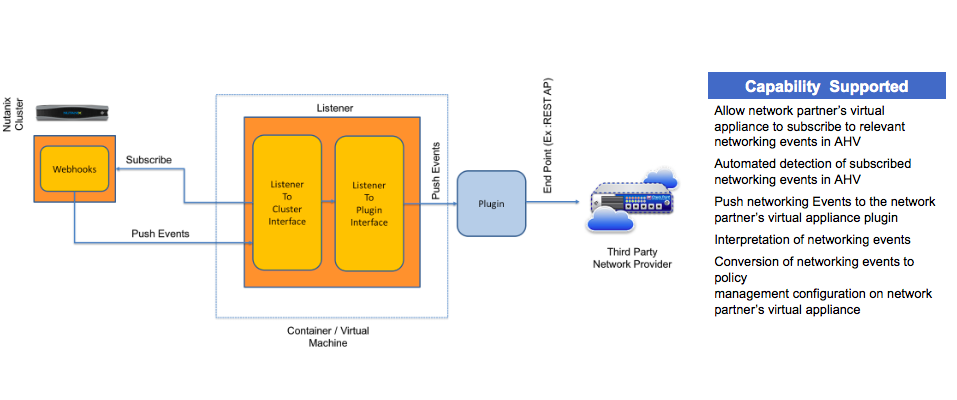
The intent of this document is to showcase on how to leverage Webhooks Listener library capability to subscribe to relevant network events occurring on AHV and help consume those events through an event consumer which will be in the form of a third party plugin.

This document uses a use case as an illustration to demonstrate the pedagogy of approach to be followed in integrating an event consumer with the listener library. The listener library is implemented in GoLang.

# Assumptions

* Consumers of this framework should have exposure to GoLang programming language
* The plugin implementation steps described are for illustration purposes only. Consumers are expected to exercise their own discretion in determining their own pedagogy of approach to write their own plugin. They can use the implementation steps for guidance.

# Listener Plugin Architecture



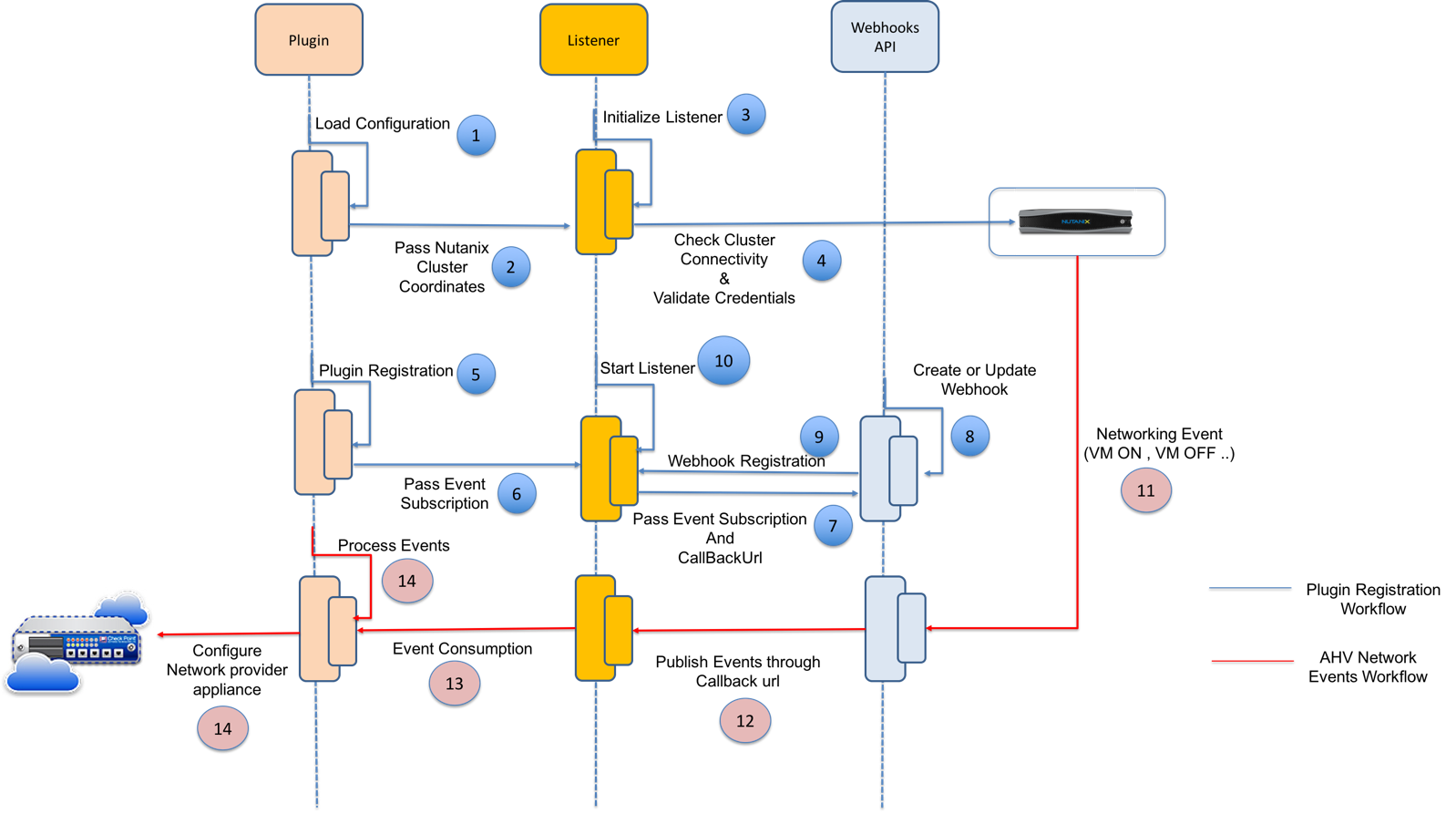
Nutanix is providing a listener-plugin framework written in GoLang which would enable third party network providers to subscribe to relevant networking events occurring on AHV. The framework will be comprised by two components (1) Listener (2) Event Consumer(plugin).

The plugin component will define the configuration file which will contain the end point details of the the network virtual appliance along with the appropriate credentials for accessing the virtual appliance through the end point. The plugin will also pass the relevant AHV networking events that it is interested in to the listener along with the callback url which the listener can use to publish the networking events to the plugin. The plugin will contain the logic on how to interpret the events as and when they are published using the callback url and make appropriate calls to the third party virtual appliance using the appropriate end point and configure it.

The listener on it’s part will register the plugin with the Nutanix AHV cluster by creating a web hook using the events and callback url information provided by the plugin. The listener will listen for AHV events and as and when the subscribed events occur, they will be made available to the plugin through the callback url.

The listener-plugin framework is open sourced. Partners can implement their own plugin using the interfaces provided by the GoLang framework. Partners can introduce new interfaces based on their requirement. Partners are also free to make changes to the listener code as deemed necessary.

# Listener Plugin Workflow



**Step-1)** Load the third party virtual appliance configuration file. The configuration file will contain the end point details of the the network virtual appliance along with the appropriate credentials for accessing the virtual appliance through the end point. The configuration may also contain other configuration details which will be contingent on the plugin implementation and that may be consumed by the same/

**Step-2)** Initialization of the listener object takes place

**Step-3)** As part of the initialization of the listener object, the listener receives the Nutanix cluster end point coordinates and credentials in the form of Prism virtual IP and Prism credentials

**Step-4)** The listener makes dummy REST calls and validates connectivity to the cluster using the Nutanix cluster end point coordinates in step-3.

**Step-5)** In this particular step, the event consumer(plugin) register itself with the Nutanix cluster. It will assign the event consumer's interface as the point of invocation on occurrence of the subscribed event.

**Step-6)** As part of the event consumer (plugin) registration, the event consumer subscribes to relevant events occurring on the cluster through Webhooks. Some of the relevant events are

"VM.CREATE", "VM.DELETE", "VM.ON", "VM.OFF", "VM.UPDATE",

"VM.MIGRATE", "VM.NIC\_PLUG", "VM.NIC\_UNPLUG"

**Step-7)** The event consumer passes the registration events and a call back url which will be used for creating Webhooks

**Step-8)** Web hook Registration process by making the appropriate REST calls to Nutanix AHV

**Step-9)** A new web hook is created or an existing web hook is updated based on the information provided in step-7

**Step-10)** Start the web hook listener. Open a HTTP socket on the listener's port & listens for event

// notifications from web hooks on the listener's callback URL.

**Step-11)** Networking event occurs on AHV

**Step-12)** The subscribed networking event details are published through the plugin callback url

**Step-13)** The plugin interprets the networking event through the implemented logic.

**Step-14)** The plugin configures the network virtual appliance by making appropriate end point call to the third party network virtual appliannce

# Listener Library and Plugin Implementation Steps and Pseudo Code

|  |  |
| --- | --- |
| ***Step-1 )*** *Create an Event Consumer(plugin) Configuration File .*   * + - 1. *consumer configuration file .* | *The event consumer(plugin) configuration file would contain the following information*  *1) Nutanix cluster connection details (Cluster External IP, Prism username and Prism password)*  *2) Third party product connection details (IP, username and password)*  *3) Relevant optional configuration parameters that will be consumed by the event consumer.*  *NOTE:*  *1) Developers should exercise their discretion in using their choice of encryption mechanism for encoding and decoding credentials.*  *2) Developers should exercise their discretion in determining the configuration parameters and source of the configuration parameters. For illustration purposes , this configuration file for F5 BIG IP virtual appliance contains the "Load Balancing Pool name*    *{*  *"<plugin\_name>\_instance\_config" :*  *{*  *"ip": "10.x.x.x",*  *"port": "<port\_number>",*  *"username": "<encoded\_username>",*  *"password": "<encoded\_password>",*  *"serviceport": "8080",*  *"pools":*  *[*  *{*  *"pool\_name": "test-pool",*  *"pool\_members": ["10.X.X.X:8080"]*  *}*  *]*  *},*  *"nutanix\_cluster\_config":*  *{*  *"ip": "10.X.X.X",*  *"port": "9440",*  *"username": "<encoded\_username>",*  *"password": "<encoded\_password>",*  *}*  *}* |

|  |  |
| --- | --- |
| ***Step-2*** *In order to enable the event consumer (plugin) to consume the parameters defined in configuration file in step-1, we need create an event consumer (plugin) schema file . Here we need to match every relevant field from the configuration file created in step-1 to a relevant data structure in the (schema) file.*  *The schema file be used by the event consumer (plugin) to pass the cluster credentials to the listener , connect to the third party product and consume the event based upon the defined optional parameters .*  *A schema file is a requirement as per GoLang .* | *Below data is GoLang programming language native data structure to which we are going to associate data defined in configuration file. The following schema file matches each parameter defined in the configuration file in Step-1 through relevant datastructures*    *package config*  *type F5Config struct*  *{*  *F5InstanceConfig F5InstanceConfig `json:"f5\_instance\_config"`*  *NutanixClusterConfig NutanixClusterConfig `json:"nutanix\_cluster\_config"`*  *}*  *type F5InstanceConfig struct*  *{*  *IP string `json:"ip"`*  *Port string `json:"port"`*  *Username string `json:"username"`*  *Password string `json:"password"`*  *ServicePort string `json:"serviceport"`*  *Pools []Pool `json:"pools"`*  *}*  *type Pool struct*  *{*  *PoolName string `json:"pool\_name"`*  *PoolMembers []string `json:"pool\_members"`*  *}*  *type NutanixClusterConfig struct*  *{*  *IP string `json:"ip"`*  *Port string `json:"port"`*  *Username string `json:"username"`*  *Password string `json:"password"`*  *}* |

|  |  |
| --- | --- |
| ***Step – 3)*** *In order to enable the listener to consume the event data that will be published by Webhooks , we need to define a schema file for the listener. Here we need to match every relevant field from the event data to a relevant data structure in the (schema) file.*  *The schema file be used by the listener to unmarshall the event data .* | *Below schema is the GoLang programming language native data structure to which we are going to associate each element of the event data published by webhooks.*  *package schema*  *// Schema definition for the webhook event.*  *type Event struct {*  *EntityReference Reference `json:"entity\_reference"`*  *Data Data `json:"data"`*  *Version string `json:"version"`*  *Event\_Type string `json:"event\_type"`*  *}*  *type Reference struct {*  *KIND string `json:"kind"`*  *UUID string `json:"uuid"`*  *}*  *type Data struct {*  *Metadata EventMetadata `json:"metadata"`*  *}*  *type EventMetadata struct {*  *Status Status `json:"status"`*  *Spec EventSpec `json:"spec"`*  *APIVersion string `json:"api\_version"`*  *Metadata EventSubMetadata `json:"metadata"`*  *}*  *type Status struct {*  *State string `json:"state"`*  *Name string `json:"name"`*  *Resources EventResources `json:"resources"`*  *}*  *type EventSpec struct {*  *}*  *type EventResources struct {*  *NICList []NIC `json:"nic\_list"`*  *HostReference HostReference `json:"host\_reference"`*  *HypervisorType string `json:"hypervisor\_type"`*  *NumVCPUsPerSocket int `json:"num\_vcpus\_per\_socket"`*  *NumSockets int `json:"num\_sockets"`*  *MemorySizeMB int `json:"memory\_size\_mb"`*  *GPUList []string `json:"gpu\_list"`*  *PowerState string `json:"power\_state"`*  *DiskList []DiskList `json:"disk\_list"`*  *}*  *type NIC struct {*  *NetworkReference NetworkReference `json:"network\_reference"`*  *MACAddress string `json:"mac\_address"`*  *}*  *type NetworkReference struct {*  *Kind string `json:"kind"`*  *UUID string `json:"uuid"`*  *}*  *type HostReference struct {*  *Kind string `json:"kind"`*  *UUID string `json:"uuid"`*  *}*  *type DiskList struct {*  *DeviceProperties DeviceProperties `json:"device\_properties"`*  *}*  *type DeviceProperties struct {*  *DiskAddress DiskAddress `json:"disk\_address"`*  *DeviceType string `json:"device\_type"`*  *}*  *type DiskAddress struct {*  *DeviceIndex int `json:"device\_index"`*  *AdapterType string `json:"adapter\_type"`*  *}*  *type EventSubMetadata struct {*  *OwnerReference OwnerReference `json:"owner\_reference"`*  *Kind string `json:"kind"`*  *EntityVersion int `json:"entity\_version"`*  *UUID string `json:"uuid"`*  *Categories Categories `json:"categories"`*  *}*  *type OwnerReference struct {*  *Kind string `json:"kind"`*  *UUID string `json:"uuid"`*  *Name string `json:"name"`*  *}*  *type Categories struct {*  *}* |
| ***Step-4)*** *Create Listener interface* | *In GoLang we are required to define interfaces for each method that will be part of the actual listener code. The event consumer will invoke the following methods*   1. *Initialize() : For listener initialization . This method will be invoked by the event consumer in order to initialize the Listener object. It will receive the Nutanix cluster details from the event consumer and initialize the listener after validating the cluster (whether the credentials are correct etc.)* 2. *RegisterForEvents() : This method allows the event consumer to register itself with the Nutanix cluster subscribing to relevant events occurring on the cluster through webhooks.*   *type Listener interface*  *{*  *Initialize(ip string, port string, username string,password string) (Listener, error)*  *RegisterForEvents(events []string, eventConsumer EventConsumer) (error)*  *}* |
| ***Step-5)*** *Implementation of listener interface for webhooks* | *The following are the implementation steps of the Webhooks listener illustrated through some sample pseudo code. The implementation approach can be broadly divided into the following steps*   1. *Import relevant packages* 2. *Implement Listener initialization through Initialize() . This method will be invoked by the event consumer in order to initialize the Listener object.*   *It will receive the Nutanix cluster details from the event consumer &*  *initialize the listener after validating the cluster details*  *(whether the credentials are correct etc.)*   1. *Implement method for creating or updating webhooks* 2. *Implement method for starting listener* 3. *Implement method which will be invoked when the listener receives an event . It will invoke the event consumer’s callback method and pass the received event to the event consumer (plugin)* 4. *Implement the method for events registration through RegisterForEvents(). This method allows the event consumer to register itself with the Nutanix cluster subscribing to relevant events occurring on the cluster through webhooks. It will assign the event consumer's interface as the point of invocation on occurrence of the subscribed event.*   *// Import relevant packages*  *import (*  *"fmt"*  *"net/http"*  *"errors"*  *"encoding/json"*  *"github.com/golang/glog"*  *"io/ioutil"*  *"reflect"*  *"webhooks/listener/webhooksschema"*  *"webhooks/listener/consts"*  *"webhooks/listener/utils"*  *"webhooks/interfaces/schema"*  *"webhooks/interfaces"*  *)*  *// Listener Initialization Implementation*  *func (webhooksListener WebhooksListener) Initialize(ip string, port string,*  *username string, password string) (WebhooksListener, error) {*  *var err error*  ***\*\*\*\*\* Pass cluster credentials to the listener \*\*\*\*\****  *webhooksListener.clusterIp = ip*  *webhooksListener.clusterPort = port*  *webhooksListener.clusterUsername = username*  *webhooksListener.clusterPassword = password*  ***\*\*\*\*\*\* Pass Listener port \*\*\*\*\*\****  *if (webhooksListener.ListenerPort == "")*  *{*  *webhooksListener.ListenerPort = consts.DefaultListenerPort*  *}*  ***\*\*\*\*\*\* Check network connectivity with cluster \*\*\*\*\*\****  *localIp, err := utils.CheckOutboundConnectivity(webhooksListener.clusterIp,*  *webhooksListener.clusterPort)*  *if (err != nil) {*  *glog.Error("Failed to verify connectivity with cluster.", err)*  *return webhooksListener, err*  *}*  ***\*\*\*\*\*\* Assign Listener IP \*\*\*\*\*\****  *webhooksListener.listenerIp = localIp*  ***\*\*\*\*\*\* Check whether credentials are valid \*\*\*\*\*\****  *webhooksListener.clusterPort, consts.GetCurrentUser)*  *request := utils.PrepareRequest(requestURL, webhooksListener.clusterUsername,*  *webhooksListener.clusterPassword, "GET")*  *glog.Info("Making http request : %v", requestURL)*  *response, err := utils.DoRequest(request)*  *if (err != nil) {*  *glog.Error("Unable to login cluster with given credentials. Error: ", err)*  *return**webhooksListener, err*  *}*  *// Implement method for creating or updating webhooks*  *func (webhooksListener WebhooksListener) createOrUpdateWebhook(*  *events []string) (error)*  ***\*\*\*\*\*\* Create Webhook name \*\*\*\*\*\****  *webhookName := fmt.Sprintf("%s%s",*  *consts.WebhookNamePrefix, webhooksListener.listenerIp)*  ***\*\*\*\*\* Create webhook url which will be used by Webhooks for publishing the events to listener \*\*\*\*\****  *RequestURL := fmt.Sprintf("https://%s:%s%s", webhooksListener.clusterIp,*  *webhooksListener.clusterPort, consts.ListWebhooks)*  ***\*\*\*\*\* Invoke REST API to get list of webhooks \*\*\*\*\****  *request:=utils.PrepareRequest(requestURL, webhooksListener.clusterUsername,*  *webhooksListener.clusterPassword, "POST")*  ***\*\*\*\* Retreive the list of Webhooks and match the list of the listener url to ascertain whether there is an existing webhook for the listener \*\*\****  *var webhookToUpdate webhooksschema.Webhook*  *for \_, webhook := range currentWebhooks.Entities {*  *glog.Info("Webhook url :", webhook.Spec.Resources.PostURL)*  *if (webhook.Spec.Resources.PostURL == postUrl) {*  *glog.Info("Found matching webhook.")*  *webhookToUpdate = webhook*  *break*  *}*  ***\*\*\* If no existing webhook found then create a new webhook or update the existing webhook \*\*\****  *if (webhookToUpdate.Metadata.UUID == "") {*  *glog.Info("No existing webhook found. Creating new webhook.")*  *requestURL = fmt.Sprintf("https://%s:%s%s", webhooksListener.clusterIp,*  *webhooksListener.clusterPort, consts.CreateWebhook)*  *requestMethod = "POST"*  *eventList = events*  *} else {*  *glog.Info("Updating existing webhook.")*  *requestURL = fmt.Sprintf("https://%s:%s%s%s", webhooksListener.clusterIp,*  *webhooksListener.clusterPort, consts.UpdateWebhook,*  *webhookToUpdate.Metadata.UUID)*  *requestMethod = "PUT"*  *eventList = append(webhookToUpdate.Spec.Resources.EventsFilterList,*  *events...)*  *eventList = utils.RemoveDuplicates(eventList)*  *}*  *// Implement method for starting listener*  *func (webhooksListener WebhooksListener) startListener() (error){*  *var err error*  *if(webhooksListener.ListenerPort == "") {*  *webhooksListener.ListenerPort = consts.DefaultListenerPort*  *}*  *webhooksListener.ListenerState <- "Starting HTTP Listener .."*  *webhooksListener.ListenerPort = ":" + webhooksListener.ListenerPort*  *http.HandleFunc(consts.ListenerCallbackURL, webhooksListener.onEvent)*  *err = http.ListenAndServe(webhooksListener.ListenerPort, nil)*  *// Implement method which will be invoked when the listener receives an event . It will invoke the event consumer’s callback method and pass the received event to the event consumer (plugin)*  *func (webhooksListener WebhooksListener) onEvent(*  *responseWriter http.ResponseWriter, request \*http.Request) {*  *var event schema.Event*  ***\*\*\*\*\*\*\*Read event data \*\*\*\*\****  *body, err := ioutil.ReadAll(request.Body)*  *if err != nil {*  *glog.Info("Error reading input request body. Cannot proceed.", err)*  *return*  *}*  *eventData := string(body)*  *glog.Info("Event data : " + eventData)*  *err = json.Unmarshal([]byte(eventData), &event)*  ***\*\*\*\*\*\*\*Get event consumer callback’s method and invoke it \*\*\*\*\****  *method := reflect.ValueOf(webhooksListener.eventConsumer).MethodByName(*  *consts.EventConsumerCallbackMethod)*  *if (method.IsValid()) {*  *glog.Info("Got event consumer method.")*  *methodArgs := make([]reflect.Value, method.Type().NumIn())*  *methodArgs[0] = reflect.ValueOf(event)*  *method.Call(methodArgs)*  *}*  ***}***  *// Implement event registration method*  *func (webhooksListener WebhooksListener) RegisterForEvents(events []string,*  *eventConsumer interfaces.EventConsumer) (error)*  ***\*\*\*\*Check port availability \*\*\*\*\****  *err = utils.CheckPortAvailability(webhooksListener.ListenerPort)*  ***\*\*\*\*Create or update webhooks \*\*\*\*\****  err = webhooksListener.createOrUpdateWebhook(events)  ***\*\*\*\*Start HTTP listener \*\*\*\*\****  *webhooksListener.eventConsumer = eventConsumer*  *go webhooksListener.startListener()*  *return err* |
| ***Step- 6)***  *Create EventConsumer (plugin) interface* | *In GoLang we are required to define interfaces for each method that will be part of the actual plugin code. The listener will invoke the relevant method defined in the interface to notify the events occurring on AHV . In the below mentioned example we need to define a package interface for the method OnEvent \*\*\*\*\*\*\*\*\*\*\**  *package interfaces*  *import "webhooks/interfaces/schema"*  *type EventConsumer interface*  *{*  *OnEvent(event schema.Event) (error)*  *}* |
| ***Step- 7)***  *EventConsumer (plugin) implementation* | *The following are the implementation steps of a F5 plugin illustrated through some sample pseudo code. The implementation approach can be broadly divided into the following steps*   1. *Import relevant packages* 2. *Define Config path* 3. *Define a method for loading config file and unmarshalling the JSON configuration data (from step-1) and convert it into the schema data (from step-2)* 4. *Define a method for processing event data which would include invoking the method of loading the config file (step-c) and then process the events data*   *Note :*  *The developers are to exercise their own discretion/style of implementing their own plugin and consume the configuration data and network event data accordingly.*  *// Import Relevant packages*  *package consumer*  *import (*  *"io/ioutil"*  *"encoding/json"*  *"fmt"*  *"eventconsumer/config"*  *"github.com/golang/glog"*  *"webhooks/listener/consts"*  *"webhooks/listener/utils"*  *"webhooks/interfaces/schema"*  *)*  *// Define Config path*  *type F5EventConsumer struct {*  *// Type that implements the EventConsumer interface.*  *}*  *const (*  *F5ConfigDir = "/opt/f5/config/"*  *}*  *// Load F5 Config file and unmarshall JSON data from the config file (from step-1) to the schema data (from step-2)*  *func LoadF5Config() (config.F5Config, error)*  *{*  *var f5Config config.F5Config*  *glog.Info("Loading config..")*  *f5ConfigPath := F5ConfigDir + "f5\_config.json"*  *// Read config file*  *f5ConfigFileContent, err := ioutil.ReadFile(f5ConfigPath)*  *if (err != nil) {*  *glog.Error("Error reading config.", err)*  *return f5Config, err*  *}*  *//unmarshall JSON data*  *err = json.Unmarshal([]byte(f5ConfigFileContent), &f5Config)*  *if (err != nil) {*  *glog.Error("Failed to unmarshal config.",err)*  *}*  *return f5Config, err*  *}*  *// Define a method for processing event data which would include invoking the method of loading the config file and then process the events data*  *func (f5EventConsumer F5EventConsumer) OnEvent(event schema.Event) (error)*  *{*  *var err error*  *glog.Info("Received event of type " + event.Event\_Type)*  *// Load F5 Config file*  *f5Config, err := LoadF5Config()*  *if (err != nil) {*  *glog.Error("Failed to load config. Cannot proceed.", err)*  *return err*  *}*  *// If event type is VM.ON , then call the F5 BIGIP virtual appliance and configure it accordingly*  *switch eventType := event.Event\_Type; eventType {*  *case consts.VM\_ON: {*  *err = onVmOn(event, f5Config)*  *}* |
| ***Step 8-)***  *Create EventConsumer (plugin) Main package* | *The following are the implementation steps of a F5 plugin main package illustrated through some sample pseudo code. The implementation approach can be broadly divided into the following steps*   1. *Import relevant packages* 2. *Define the networking events that will be consumed by the event consumer* 3. *Load the event consumer configuration file* 4. *Initialize listener and pass Nutanix cluster connection details to the listener* 5. *Register for networking events* 6. *Initiate listener and listen for events*   *Note :*  *The developers are to exercise their own discretion/style of implementing their own plugin and consume the configuration data and network event data accordingly.*  *// Import relevant packages*  ***package main***  *import (*  *"eventconsumer/impl"*  *"flag"*  *"fmt"*  *"github.com/golang/glog"*  *"os"*  *…*  *…*  *}*  *// Define the networking events that will be consumed by the event consumer*  *func main() {*  *var webhooksListener listener.WebhooksListener*  *events := []string{consts.VM\_ON, consts.VM\_OFF*  *// Load the event consumer configuration file*  *f5Config, err := consumer.LoadF5Config()*  *if (err != nil) {*  *glog.Errorf("Failed to load config. Cannot proceed. Error:- %v", err)*  *return*  *}*  *// Initialize listener and pass Nutanix cluster connection details to the listener*  *webhooksListener, err = webhooksListener.Initialize(*  *f5Config.NutanixClusterConfig.IP,*  *f5Config.NutanixClusterConfig.Port,*  *f5Config.NutanixClusterConfig.Username,*  *f5Config.NutanixClusterConfig.Password)*  *// Register for networking events*  *webhooksListener.RegisterForEvents(events,consumer.F5EventConsumer{})*  *// Initiate listener and listen for events*  *listenerStateMsg, listenerRunning := <-webhooksListener.ListenerState*  *for listenerRunning == true {*    *if(listenerStateMsg != "") {*  *glog.Info("Message from Listener: ", listenerStateMsg)*  *}*  *listenerStateMsg, listenerRunning = <-webhooksListener.ListenerState*  *}*  *}* |

# Deploy and Run

Once Plugin development is complete using listener, follow below steps to deploy and run.

You can deploy your own user VMor container and run the listener and plugin there.

**Deploy:**

Please leverage your own deployment GoLang deployment environment to build and create a binary once the development is complet .

You can refer this link : <https://golang.org/doc/code.html#Command>

**Run Go Code:**

Deployment will give you binary generated from build process. You can run that binary as per below command.

Example : ./<binary\_file\_name> -stderrthreshold =info

**./f5eventconsumer -stderrthreshold=INFO**

stderrthreshold is input flag which allows the verbose rendering of the execution

# Sample Logs

2017/02/27 21:48:40 Loading config..

2017/02/27 21:48:40 Initializing listener..

2017/02/27 21:48:40 Verifying connectivity with cluster.

2017/02/27 21:48:40 Checking connectivity with 10.5.136.203:9440

2017/02/27 21:48:40 Connectivity successfully verified using local IP 10.4.66.146.

2017/02/27 21:48:40 Checking if the credentials are valid..

2017/02/27 21:48:40 Verifying against : https://10.5.136.203:9440/api/nutanix/v3/users/me

2017/02/27 21:48:40 Processing request : https://10.5.136.203:9440/api/nutanix/v3/users/me

2017/02/27 21:48:40 Request successful. 200

2017/02/27 21:48:40 Registering for events [VM.ON VM.OFF]

2017/02/27 21:48:40 Checking if port 8080 is available.

2017/02/27 21:48:40 Getting existing webhooks..

2017/02/27 21:48:40 Request data : {"kind":"webhook"}

2017/02/27 21:48:40 Request URL : https://10.5.136.203:9440/api/nutanix/v3/webhooks/list

2017/02/27 21:48:40 Processing request : https://10.5.136.203:9440/api/nutanix/v3/webhooks/list

2017/02/27 21:48:41 Request successful. 200

2017/02/27 21:48:41 Total existing webhooks: 2

2017/02/27 21:48:41 Looking for webhook with name : Nutanix\_Listener\_Webhook\_10.4.66.146

2017/02/27 21:48:41 Webhook name : Nutanix\_Listener\_Webhook\_10.4.66.146

2017/02/27 21:48:41 Found matching webhook.

2017/02/27 21:48:41 Updating existing webhook.

2017/02/27 21:48:41 Request data : {"metadata":{"kind":"webhook"},"spec":{"name":"Nutanix\_Listener\_Webhook\_10.4.66.146","resources":{"post\_url":"http://10.4.66.146:8080/listener/callback","events\_filter\_list":["VM.ON","VM.OFF"]},"description":""},"api\_version":"3.0"}

2017/02/27 21:48:41 Request URL : https://10.5.136.203:9440/api/nutanix/v3/webhooks/9dd95b78-3880-48be-a6dd-43204d51d691

2017/02/27 21:48:41 Processing request : https://10.5.136.203:9440/api/nutanix/v3/webhooks/9dd95b78-3880-48be-a6dd-43204d51d691

2017/02/27 21:48:41 Request successful. 200

2017/02/27 21:48:41 Successfully completed webhook operation.

2017/02/27 21:48:41 Starting HTTP Listener ..

2017/02/27 21:49:16 Received event.

2017/02/27 21:49:16 Event data : {"entity\_reference": {"kind": "vm", "uuid": "96d9e5aa-f423-4356-a4ff-5c989ac5d502"}, "data": {"metadata": {"status": {"state": "kComplete", "name": "test-vm-01", "resources": {"nic\_list": [], "host\_reference": {"kind": "host", "uuid": "29c149a3-0118-4c80-a029-29ab9e07e885"}, "hypervisor\_type": "AHV", "num\_vcpus\_per\_socket": 1, "num\_sockets": 4, "memory\_size\_mb": 1024, "gpu\_list": [], "power\_state": "POWERED\_ON", "disk\_list": [{"device\_properties": {"disk\_address": {"device\_index": 0, "adapter\_type": "IDE"}, "device\_type": "CDROM"}}]}}, "spec": {}, "api\_version": "3.0", "metadata": {"owner\_reference": {"kind": "user", "uuid": null, "name": null}, "kind": "vm", "entity\_version": 1, "uuid": "96d9e5aa-f423-4356-a4ff-5c989ac5d502", "categories": {}}}}, "version": "1.0", "event\_type": "VM.ON"}

2017/02/27 21:49:16 Got event consumer method.

2017/02/27 21:49:16 Received event of type VM.ON

2017/02/27 21:49:16 Loading config..

2017/02/27 21:49:16 Processing VM.ON event.

2017/02/27 21:49:16 https://10.5.138.7:443/mgmt/tm/ltm/pool/test-pool/members

2017/02/27 21:49:16 {"name": "10.5.4.2:8080"}

2017/02/27 21:49:16 Processing request : https://10.5.138.7:443/mgmt/tm/ltm/pool/test-pool/members

2017/02/27 21:49:17 Request successful. 200

2017/02/27 21:49:17 Successfully added member to pool. {"kind":"tm:ltm:pool:members:membersstate","name":"10.5.4.2:8080","partition":"Common","fullPath":"/Common/10.5.4.2:8080","generation":975,"selfLink":"https://localhost/mgmt/tm/ltm/pool/test-pool/members/~Common~10.5.4.2:8080?ver=12.1.1","address":"10.5.4.2","connectionLimit":0,"dynamicRatio":1,"ephemeral":"false","fqdn":{"autopopulate":"disabled"},"inheritProfile":"enabled","logging":"disabled","monitor":"default","priorityGroup":0,"rateLimit":"disabled","ratio":1,"session":"user-enabled","state":"unchecked"}