

개방형 클라우드 플랫폼 기술지원 및 유지보수·관리

Rep Guide (v5.1.0-dev)

모니터링

|  |  |  |
| --- | --- | --- |
| 작성자 | 오한수 (인) | 2020.12.21 |
| 검토자 | 김태진 (인) | 2020.12.21 |
| 승인자 | 정기봉 (인) | 2020.12.21 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

개 정 이 력

| 버전 | 작성일 | 변경내용[[1]](#footnote-1) | 작성자 | 검토자 | 승인자 |
| --- | --- | --- | --- | --- | --- |
| 0.1 | 2020.11.23 | 최초작성 | 오한수 | 김태진 | 정기봉 |
| 0.2 | 2020.12.21 | 소스코드 추가 | 오한수 | 김태진 | 정기봉 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**목차**

[1. Goland 설치 4](#_Toc59459875)

[2. rep 다운로드 및 환경 구축 4](#_Toc59459876)

[1) rep 다운로드 4](#_Toc59459877)

[2) rep 환경 구축 5](#_Toc59459878)

[3) rep IDE 실행 7](#_Toc59459879)

[3. rep 빌드 9](#_Toc59459880)

[1) rep 빌드 9](#_Toc59459881)

[2) 빌드 후 바이너리 실행 9](#_Toc59459882)

[4. rep 테스트 10](#_Toc59459883)

[1) 실 서버 테스트 (rep 바이너리 파일 위치 및 실행 ) 10](#_Toc59459884)

[○ paasta deployment 및 diego-cell vm 확인 10](#_Toc59459885)

[○ diego-cell 접속 10](#_Toc59459886)

[○ rep 바이너리 파일 위치 11](#_Toc59459887)

[○ rep 프로세스 확인 11](#_Toc59459888)

[○ rep Custom API 12](#_Toc59459889)

[5. rep 수정사항 13](#_Toc59459890)

[1) 수정사항 13](#_Toc59459891)

[6. rep 히스토리 13](#_Toc59459892)

[1) 테스트 13](#_Toc59459893)

[2) 소스코드 추가 14](#_Toc59459894)

[○ rep/main.go 14](#_Toc59459895)

[○ rep/routes.go 23](#_Toc59459896)

[○ rep/handler.go 24](#_Toc59459897)

[○ rep/container\_list\_handler.go 26](#_Toc59459898)

[○ executor/initializer.go 33](#_Toc59459899)

### **1. Goland 설치**

### **2. rep 다운로드 및 환경 구축**

#### 1) rep 다운로드

|  |
| --- |
| ○ Git 정보  - 주소 : <https://github.com/PaaS-TA/PaaS-TA-Monitoring/tree/v5.1.0-dev>)  - 버전 : V5.1.0-dev  - cadvisor path : paasta-agents > rep  $ git clone -b v5.1.0-dev <https://github.com/PaaS-TA/PaaS-TA-Monitoring.git> |

#### 2) rep 환경 구축

|  |
| --- |
| ○ Goland > Open Project 선택  $ ~/PaaS-TA-Monitoring/paasta-agents/rep    ○ File > Settings 선택  - Go > GOROOT 설정 (Download Go SDK를 이용.)    - Go > GOPATH 설정  rep > src > \*.zip 파일의 압축을 푼다. |

#### 3) rep IDE 실행

|  |
| --- |
| ○ Navigation > Edit Configurations… 선택    ○ Edit Configurations 설정    ○ Edit Configurations 값  - Name : 데몬 실행 이름 (**rep**)  - Run kind : 실행 종류 (**Package**)  - Package Path : 실제 main.go 존재 경로 (**code.cloudfoundry.org/rep/cmd/rep**)  - Output directory : 빌드 후 바이너리 파일 경로  (**~/PaaS-TA-Monitoring/paasta-agents/rep/bin**)  - Working Directory : 실제 rep최상위 경로  (**~/PaaS-TA-Monitoring/paasta-agents/rep**)  ○ OK 선택  ○ Debug 실행 및 결과 |

### **3. rep 빌드**

#### 1) rep 빌드

|  |
| --- |
| ○ Run 이용    - IDE Run 실행 시 rep 바이너리 파일 생성 시 과정 및 결과 |

#### 2) 빌드 후 바이너리 실행

|  |
| --- |
| ○ rep 바이너리 파일 위치 및 실행 (실제 configFilePath 지정이 필요함.)  $ cd ~/PaaS-TA-Monitoring/paasta-agents/rep  $ ./rep |

### 

### **4. rep 테스트**

#### 1) 실 서버 테스트 (rep 바이너리 파일 위치 및 실행 )

|  |
| --- |
| ○ paasta deployment 및 diego-cell vm 확인 $ bosh -e micro-bosh vms   ○ diego-cell 접속 $ bosh -e micro-bosh -d paasta ssh diego-cell/11c87e67-7c4f-4831-ba7d-b84bef38683c   ○ rep 바이너리 파일 위치 $ cd /var/vcap/packages/rep/   ○ rep 프로세스 확인 $ sudo ps -ef | grep rep   ○ rep Custom API $ curl 127.0.0.1:8080/healthz |

### **5. rep 수정사항**

#### 1) 수정사항

|  |
| --- |
| ○ rep Custom API 추가 및 수정 파일 목록  - src > code.cloudfoundry.org > rep > routes.go  - src > code.cloudfoundry.org > executor > initializer.go  - src > code.cloudfoundry.org > rep > main.go  - src > code.cloudfoundry.org > rep > handler > handler.go  - src > code.cloudfoundry.org > rep > handler > container\_list\_handler.go   1. CNI 활성, 비활성 분기 처리 2. Host Port 및 Container Port 추가 로직 버그 수정 |

### **6. rep 히스토리**

#### 1) 테스트

|  |
| --- |
| /var/vcap/jobs/cadvisor/config/certs/rep  cd /var/vcap/jobs/cadvisor/config/certs/rep/  curl -k --cacert client.crt --cert client.crt --key client.key <https://127.0.0.1:1800/v1/containers>  select count(\*) from cpu\_usage\_per\_cpu;  curl -k --cacert /var/vcap/jobs/cadvisor/config/certs/rep/client.crt --cert /var/vcap/jobs/cadvisor/config/certs/rep/client.crt --key /var/vcap/jobs/cadvisor/config/certs/rep/client.key <https://127.0.0.1:1800/v1/containers>  [  {  "limits": {  "fds": 16384,  "mem": 1024,  "disk": 1024  },  "usage\_metrics": {  "memory\_usage\_in\_bytes": 291344384,  "disk\_usage\_in\_bytes": 187248640,  "time\_spent\_in\_cpu": 95023342183  },  "container\_id": "4536d9c7-3483-4d23-44e1-f945",  "interface\_id": "wfbsk3eo1lg1-0",  "application\_id": "15ff0a4e-7107-44ad-9ece-40b3833e6c40",  "application\_index": "0",  "application\_name": "spring-music-pinpoint",  "application\_uris": [  "spring-music-pinpoint-nice-impala-qb.182.252.135.97.xip.io"  ]  }  ] |

#### 2) 소스코드 추가

(// Adde for PaaS-TA 주석 참고)

|  |
| --- |
| ○ rep/main.go package main  import (  "crypto/tls"  "crypto/x509"  "encoding/json"  "encoding/pem"  "errors"  "flag"  "fmt"  "io/ioutil"  "net"  "net/http"  "os"  "strings"  "time"   "code.cloudfoundry.org/bbs"  "code.cloudfoundry.org/bbs/models"  "code.cloudfoundry.org/cfhttp"  "code.cloudfoundry.org/clock"  "code.cloudfoundry.org/consuladapter"  "code.cloudfoundry.org/debugserver"  loggingclient "code.cloudfoundry.org/diego-logging-client"  "code.cloudfoundry.org/executor"  executorinit "code.cloudfoundry.org/executor/initializer"  "code.cloudfoundry.org/go-loggregator/runtimeemitter"  "code.cloudfoundry.org/lager"  "code.cloudfoundry.org/lager/lagerflags"  "code.cloudfoundry.org/localip"  "code.cloudfoundry.org/locket"  "code.cloudfoundry.org/locket/lock"  "code.cloudfoundry.org/locket/metrics/helpers"  locketmodels "code.cloudfoundry.org/locket/models"  "code.cloudfoundry.org/operationq"  "code.cloudfoundry.org/rep"  "code.cloudfoundry.org/rep/auctioncellrep"  "code.cloudfoundry.org/rep/cmd/rep/config"  "code.cloudfoundry.org/rep/evacuation"  "code.cloudfoundry.org/rep/evacuation/evacuation\_context"  "code.cloudfoundry.org/rep/generator"  "code.cloudfoundry.org/rep/handlers"  "code.cloudfoundry.org/rep/harmonizer"  "code.cloudfoundry.org/rep/maintain"  "code.cloudfoundry.org/tlsconfig"  "github.com/hashicorp/consul/api"  uuid "github.com/nu7hatch/gouuid"  "github.com/tedsuo/ifrit"  "github.com/tedsuo/ifrit/grouper"  "github.com/tedsuo/ifrit/sigmon"  "github.com/tedsuo/rata"   // Adde for PaaS-TA  GardenClient "code.cloudfoundry.org/garden/client" )  var configFilePath = flag.String(  "config",  "",  "The path to the JSON configuration file.", )  var zoneOverride = flag.String(  "zone",  "",  "The availability zone associated with the rep. This overrides the zone value in the config file, if specified.", )  func main() {  flag.Parse()   repConfig, err := config.NewRepConfig(\*configFilePath)  if err != nil {  panic(err.Error())  }   if \*zoneOverride != "" {  repConfig.Zone = \*zoneOverride  }   // We need to keep this here since dockerdriver still uses cfhttp v1  cfhttp.Initialize(time.Duration(repConfig.CommunicationTimeout))   clock := clock.NewClock()  logger, reconfigurableSink := lagerflags.NewFromConfig(repConfig.SessionName, repConfig.LagerConfig)   if !repConfig.ExecutorConfig.Validate(logger) {  logger.Fatal("", errors.New("failed-to-configure-executor"))  }   if repConfig.CellID == "" {  logger.Error("invalid-cell-id", errors.New("-cellID must be specified"))  os.Exit(1)  }   metronClient, err := initializeMetron(logger, repConfig)  if err != nil {  logger.Error("failed-to-initialize-metron-client", err)  os.Exit(1)  }   rootFSMap := repConfig.PreloadedRootFS.StackPathMap()   // Adde for PaaS-TA  //executorClient, containerMetricsProvider, executorMembers, err := executorinit.Initialize(logger, repConfig.ExecutorConfig, repConfig.CellID, repConfig.Zone, rootFSMap, metronClient, clock)  executorClient, containerMetricsProvider, executorMembers, gardenClient, err := executorinit.Initialize(logger, repConfig.ExecutorConfig, repConfig.CellID, repConfig.Zone, rootFSMap, metronClient, clock)  if err != nil {  logger.Error("failed-to-initialize-executor", err)  os.Exit(1)  }  defer executorClient.Cleanup(logger)   consulClient := initializeConsulClient(logger, repConfig)   serviceClient := maintain.NewCellPresenceClient(consulClient, clock)   evacuatable, evacuationReporter, evacuationNotifier := evacuation\_context.New()   // only one outstanding operation per container is necessary  queue := operationq.NewSlidingQueue(1)   evacuator := evacuation.NewEvacuator(  logger,  clock,  executorClient,  evacuationNotifier,  repConfig.CellID,  time.Duration(repConfig.EvacuationTimeout),  time.Duration(repConfig.EvacuationPollingInterval),  )   bbsClient := initializeBBSClient(logger, repConfig)  url := repURL(repConfig)  address := repAddress(logger, repConfig)  cellPresence := initializeCellPresence(address, serviceClient, executorClient, logger, repConfig, repConfig.PreloadedRootFS.Names(), url)  batchContainerAllocator := auctioncellrep.NewContainerAllocator(auctioncellrep.GenerateGuid, rootFSMap, executorClient)  auctionCellRep := auctioncellrep.New(  repConfig.CellID,  repConfig.CellIndex,  url,  rootFSMap,  containerMetricsProvider,  repConfig.SupportedProviders,  repConfig.Zone,  executorClient,  evacuationReporter,  repConfig.PlacementTags,  repConfig.OptionalPlacementTags,  repConfig.ProxyMemoryAllocationMB,  repConfig.EnableContainerProxy,  batchContainerAllocator,  )   requestTypes := []string{  "State", "ContainerMetrics", "Perform", "Reset", "StopLRPInstance", "CancelTask", //over https only  }  requestMetrics := helpers.NewRequestMetricsNotifier(logger, clock, metronClient, time.Duration(repConfig.ReportInterval), requestTypes)  // Adde for PaaS-TA  //httpServer := initializeServer(auctionCellRep, executorClient, evacuatable, requestMetrics, logger, repConfig, false)  //httpsServer := initializeServer(auctionCellRep, executorClient, evacuatable, requestMetrics, logger, repConfig, true)  httpServer := initializeServer(auctionCellRep, executorClient, gardenClient, evacuatable, requestMetrics, logger, repConfig, false)  httpsServer := initializeServer(auctionCellRep, executorClient, gardenClient, evacuatable, requestMetrics, logger, repConfig, true)    opGenerator := generator.New(  repConfig.CellID,  rootFSMap,  repConfig.LayeringMode,  bbsClient,  executorClient,  metronClient,  evacuationReporter,  )   cleanup := evacuation.NewEvacuationCleanup(  logger,  repConfig.CellID,  time.Duration(repConfig.GracefulShutdownInterval),  time.Duration(repConfig.ExecutorConfig.EnvoyConfigReloadDuration),  bbsClient,  executorClient,  clock,  metronClient,  )   \_, portString, err := net.SplitHostPort(repConfig.ListenAddr)  if err != nil {  logger.Fatal("failed-invalid-server-address", err)  }  portNum, err := net.LookupPort("tcp", portString)  if err != nil {  logger.Fatal("failed-invalid-server-port", err)  }   bulker := harmonizer.NewBulker(  logger,  time.Duration(repConfig.PollingInterval),  time.Duration(repConfig.EvacuationPollingInterval),  evacuationNotifier,  clock,  opGenerator,  queue,  metronClient,  )   members := grouper.Members{  {"presence", cellPresence},  {"http\_server", httpServer},  {"https\_server", httpsServer},  {"evacuation-cleanup", cleanup},  {"bulker", bulker},  {"event-consumer", harmonizer.NewEventConsumer(logger, opGenerator, queue)},  {"evacuator", evacuator},  {"request-metrics-notifier", requestMetrics},  }   if repConfig.EnableConsulServiceRegistration {  registrationRunner := initializeRegistrationRunner(logger, consulClient, repConfig, portNum, clock)  members = append(members, grouper.Member{"registration-runner", registrationRunner})  }   members = append(executorMembers, members...)   if repConfig.DebugAddress != "" {  members = append(grouper.Members{  {"debug-server", debugserver.Runner(repConfig.DebugAddress, reconfigurableSink)},  }, members...)  }   group := grouper.NewOrdered(os.Interrupt, members)   monitor := ifrit.Invoke(sigmon.New(group))   logger.Info("started", lager.Data{"cell-id": repConfig.CellID})   err = <-monitor.Wait()  if err != nil {  logger.Error("exited-with-failure", err)  os.Exit(1)  }   logger.Info("exited") }  func initializeCellPresence(  address string,  serviceClient maintain.CellPresenceClient,  executorClient executor.Client,  logger lager.Logger,  repConfig config.RepConfig,  preloadedRootFSes []string,  repUrl string, ) ifrit.Runner {  if repConfig.CellRegistrationsLocketEnabled {  locketClient, err := locket.NewClient(logger, repConfig.ClientLocketConfig)  if err != nil {  logger.Fatal("failed-to-construct-locket-client", err)  }   guid, err := uuid.NewV4()  if err != nil {  logger.Fatal("failed-to-generate-guid", err)  }   resources, err := executorClient.TotalResources(logger)  if err != nil {  logger.Fatal("failed-to-get-total-resources", err)  }  cellCapacity := models.NewCellCapacity(int32(resources.MemoryMB), int32(resources.DiskMB), int32(resources.Containers))  cellPresence := models.NewCellPresence(repConfig.CellID, address, repUrl,  repConfig.Zone, cellCapacity, repConfig.SupportedProviders,  preloadedRootFSes, repConfig.PlacementTags, repConfig.OptionalPlacementTags)   payload, err := json.Marshal(cellPresence)  if err != nil {  logger.Fatal("failed-to-encode-cell-presence", err)  }   lockPayload := &locketmodels.Resource{  Key: repConfig.CellID,  Owner: guid.String(),  Value: string(payload),  TypeCode: locketmodels.PRESENCE,  Type: locketmodels.PresenceType,  }   logger.Debug("presence-payload", lager.Data{"payload": lockPayload})  return lock.NewPresenceRunner(  logger,  locketClient,  lockPayload,  int64(time.Duration(repConfig.LockTTL)/time.Second),  clock.NewClock(),  locket.RetryInterval,  )  } else {  config := maintain.Config{  CellID: repConfig.CellID,  RepAddress: address,  RepUrl: repUrl,  Zone: repConfig.Zone,  RetryInterval: time.Duration(repConfig.LockRetryInterval),  RootFSProviders: repConfig.SupportedProviders,  PreloadedRootFSes: preloadedRootFSes,  PlacementTags: repConfig.PlacementTags,  OptionalPlacementTags: repConfig.OptionalPlacementTags,  }   return maintain.New(  logger,  config,  executorClient,  serviceClient,  time.Duration(repConfig.LockTTL),  clock.NewClock(),  )  } }  func initializeServer(  auctionCellRep \*auctioncellrep.AuctionCellRep,  executorClient executor.Client,  // Adde for PaaS-TA  gardenClient GardenClient.Client,  evacuatable evacuation\_context.Evacuatable,  requestMetrics helpers.RequestMetrics,  logger lager.Logger,  repConfig config.RepConfig,  networkAccessible bool, ) ifrit.Runner {  // Adde for PaaS-TA  //handlers := handlers.New(auctionCellRep, auctionCellRep, executorClient, evacuatable, requestMetrics, logger, networkAccessible)  handlers := handlers.New(auctionCellRep, auctionCellRep, gardenClient, executorClient, evacuatable, requestMetrics, logger, networkAccessible)  routes := rep.NewRoutes(networkAccessible)  router, err := rata.NewRouter(routes, handlers)   if err != nil {  logger.Fatal("failed-to-construct-router", err)  }   listenAddress := repConfig.ListenAddr  if networkAccessible {  listenAddress = repConfig.ListenAddrSecurable  }   if !networkAccessible {  err = verifyCertificate(repConfig.CertFile)  if err != nil {  logger.Fatal("tls-configuration-failed", err)  }  }   tlsConfig, err := tlsconfig.Build(  tlsconfig.WithInternalServiceDefaults(),  tlsconfig.WithIdentityFromFile(repConfig.CertFile, repConfig.KeyFile),  ).Server(tlsconfig.WithClientAuthenticationFromFile(repConfig.CaCertFile))  if err != nil {  logger.Fatal("tls-configuration-failed", err)  }  return startTLSServer(listenAddress, router, tlsConfig) }  func startTLSServer(addr string, handler http.Handler, tlsConfig \*tls.Config) ifrit.Runner {  return ifrit.RunFunc(func(signals <-chan os.Signal, ready chan<- struct{}) error {  listener, err := net.Listen("tcp", addr)  if err != nil {  return err  }  listener = tls.NewListener(listener, tlsConfig)  close(ready)  go http.Serve(listener, handler)  <-signals  return listener.Close()  }) }  func initializeBBSClient(  logger lager.Logger,  repConfig config.RepConfig, ) bbs.InternalClient {  bbsClient, err := bbs.NewClientWithConfig(bbs.ClientConfig{  URL: repConfig.BBSAddress,  IsTLS: true,  CAFile: repConfig.CaCertFile,  CertFile: repConfig.CertFile,  KeyFile: repConfig.KeyFile,  ClientSessionCacheSize: repConfig.BBSClientSessionCacheSize,  MaxIdleConnsPerHost: repConfig.BBSMaxIdleConnsPerHost,  RequestTimeout: time.Duration(repConfig.CommunicationTimeout),  })  if err != nil {  logger.Fatal("failed-to-configure-secure-BBS-client", err)  }  return bbsClient }  func initializeConsulClient(  logger lager.Logger,  repConfig config.RepConfig, ) consuladapter.Client {  var consulClient consuladapter.Client   scheme, \_, err := consuladapter.Parse(repConfig.ConsulCluster)  if err != nil {  logger.Fatal("parse-consul-cluster-failed", err)  }   if scheme == "https" {  consulClient, err = consuladapter.NewTLSClientFromUrl(  repConfig.ConsulCluster,  repConfig.ConsulCACert,  repConfig.ConsulClientCert,  repConfig.ConsulClientKey,  )  } else {  consulClient, err = consuladapter.NewClientFromUrl(repConfig.ConsulCluster)  }   if err != nil {  logger.Fatal("new-client-failed", err)  }   return consulClient }  func repHost(cellID string) string {  return strings.Replace(cellID, "\_", "-", -1) }  func repBaseHostName(advertiseDomain string) string {  return strings.Split(advertiseDomain, ".")[0] }  func repURL(config config.RepConfig) string {  port := strings.Split(config.ListenAddrSecurable, ":")[1]  return fmt.Sprintf("https://%s.%s:%s", repHost(config.CellID), config.AdvertiseDomain, port) }  func repAddress(logger lager.Logger, config config.RepConfig) string {  ip, err := localip.LocalIP()  if err != nil {  logger.Fatal("failed-to-fetch-ip", err)  }   listenAddress := config.ListenAddr  port := strings.Split(listenAddress, ":")[1]  return fmt.Sprintf("http://%s:%s", ip, port) }  func initializeRegistrationRunner(  logger lager.Logger,  consulClient consuladapter.Client,  repConfig config.RepConfig,  port int,  clock clock.Clock, ) ifrit.Runner {  registration := &api.AgentServiceRegistration{  Name: repBaseHostName(repConfig.AdvertiseDomain),  Port: port,  Check: &api.AgentServiceCheck{  TTL: "3s",  },  Tags: []string{repHost(repConfig.CellID)},  }  return locket.NewRegistrationRunner(logger, registration, consulClient, locket.RetryInterval, clock) }  func initializeMetron(logger lager.Logger, repConfig config.RepConfig) (loggingclient.IngressClient, error) {  client, err := loggingclient.NewIngressClient(repConfig.LoggregatorConfig)  if err != nil {  return nil, err  }   if repConfig.LoggregatorConfig.UseV2API {  emitter := runtimeemitter.NewV1(client)  go emitter.Run()  }   return client, nil }  func verifyCertificate(serverCertFile string) error {  certBytes, err := ioutil.ReadFile(serverCertFile)  if err != nil {  return err  }   var blocks []byte  certBytes = []byte(strings.TrimSpace(string(certBytes)))  for {  var block \*pem.Block  block, certBytes = pem.Decode(certBytes)  if block == nil {  return fmt.Errorf("failed parsing cert")  }  blocks = append(blocks, block.Bytes...)  if len(certBytes) == 0 {  break  }  }   certs, err := x509.ParseCertificates(blocks)  if err != nil {  return fmt.Errorf("failed parsing cert: %s", err)  }   for \_, ip := range certs[0].IPAddresses {  if ip.Equal(net.ParseIP("127.0.0.1")) {  return nil  }  }   return errors.New("invalid SAN metadata. certificate needs to contain 127.0.0.1 for IP SAN metadata.") } |

|  |
| --- |
| ○ rep/routes.go package rep  import "github.com/tedsuo/rata"  const (  *StateRoute* = "STATE"  *ContainerMetricsRoute* = "ContainerMetrics"  *PerformRoute* = "PERFORM"   *StopLRPInstanceRoute* = "StopLRPInstance"  *CancelTaskRoute* = "CancelTask"   *SimResetRoute* = "RESET"   *PingRoute* = "Ping"  *EvacuateRoute* = "Evacuate"    // Adde for PaaS-TA  *ContainerListRoute* = "ContainerList" )  func NewRoutes(networkAccessible bool) rata.Routes {  var routes rata.Routes   if networkAccessible {  routes = append(routes,  rata.Route{Path: "/state", Method: "GET", Name: *StateRoute*},  rata.Route{Path: "/container\_metrics", Method: "GET", Name: *ContainerMetricsRoute*},  rata.Route{Path: "/work", Method: "POST", Name: *PerformRoute*},   rata.Route{Path: "/v1/lrps/:process\_guid/instances/:instance\_guid/stop", Method: "POST", Name: *StopLRPInstanceRoute*},  rata.Route{Path: "/v1/tasks/:task\_guid/cancel", Method: "POST", Name: *CancelTaskRoute*},   rata.Route{Path: "/sim/reset", Method: "POST", Name: *SimResetRoute*},  )  } else {  routes = append(routes,  rata.Route{Path: "/ping", Method: "GET", Name: *PingRoute*},  rata.Route{Path: "/evacuate", Method: "POST", Name: *EvacuateRoute*},  // Adde for PaaS-TA  rata.Route{Path: "/v1/containers", Method:"GET", Name:*ContainerListRoute*},  )  }  return routes  }  var RoutesLocalhostOnly = NewRoutes(false) var RoutesNetworkAccessible = NewRoutes(true) var Routes = append(RoutesLocalhostOnly, RoutesNetworkAccessible...) |

|  |
| --- |
| ○ rep/handler.go package handlers  import (  "net/http"   "code.cloudfoundry.org/executor"  "code.cloudfoundry.org/lager"  "code.cloudfoundry.org/locket/metrics/helpers"  "code.cloudfoundry.org/rep"  "code.cloudfoundry.org/rep/auctioncellrep"  "code.cloudfoundry.org/rep/evacuation/evacuation\_context"  "github.com/tedsuo/rata"  // Adde for PaaS-TA  GardenClient "code.cloudfoundry.org/garden/client" )  func New(  localCellClient auctioncellrep.AuctionCellClient,  localMetricCollector MetricCollector,   // Adde for PaaS-TA  gardenClient GardenClient.Client,   executorClient executor.Client,  evacuatable evacuation\_context.Evacuatable,  requestMetrics helpers.RequestMetrics,  logger lager.Logger,  secure bool, ) rata.Handlers {   handlers := rata.Handlers{}  if secure {  stateHandler := newStateHandler(localCellClient, requestMetrics)  containerMetricsHandler := newContainerMetricsHandler(localMetricCollector, requestMetrics)  performHandler := newPerformHandler(localCellClient, requestMetrics)  resetHandler := newResetHandler(localCellClient, requestMetrics)  stopLrpHandler := NewStopLRPInstanceHandler(executorClient, requestMetrics)  cancelTaskHandler := newCancelTaskHandler(executorClient, requestMetrics)   handlers[rep.StateRoute] = logWrap(stateHandler.ServeHTTP, logger)  handlers[rep.ContainerMetricsRoute] = logWrap(containerMetricsHandler.ServeHTTP, logger)  handlers[rep.PerformRoute] = logWrap(performHandler.ServeHTTP, logger)  handlers[rep.SimResetRoute] = logWrap(resetHandler.ServeHTTP, logger)   handlers[rep.StopLRPInstanceRoute] = logWrap(stopLrpHandler.ServeHTTP, logger)  handlers[rep.CancelTaskRoute] = logWrap(cancelTaskHandler.ServeHTTP, logger)  } else {  pingHandler := newPingHandler(requestMetrics)  evacuationHandler := newEvacuationHandler(evacuatable, requestMetrics)   handlers[rep.PingRoute] = logWrap(pingHandler.ServeHTTP, logger)  handlers[rep.EvacuateRoute] = logWrap(evacuationHandler.ServeHTTP, logger)   // Adde for PaaS-TA  containerHandler := NewContainerListHandler(logger, executorClient, gardenClient)  handlers[rep.ContainerListRoute] = logWrap(containerHandler.ServeHTTP, logger)  }   return handlers }  // this isn't being used in the Rep anymore. It is used in tests that run a // fake cell. Without this function those tests will have to replicate the code // below. Those places are auctioneer fake\_cell\_test.go and rep's // handlers\_suite\_test.go func NewLegacy(  localCellClient auctioncellrep.AuctionCellClient,  localMetricCollector MetricCollector,   // Adde for PaaS-TA  gardenClient GardenClient.Client,   executorClient executor.Client,  evacuatable evacuation\_context.Evacuatable,  requestMetrics helpers.RequestMetrics,  logger lager.Logger, ) rata.Handlers {  // Adde for PaaS-TA  //insecureHandlers := New(localCellClient, localMetricCollector, executorClient, evacuatable, requestMetrics, logger, false)  //secureHandlers := New(localCellClient, localMetricCollector, executorClient, evacuatable, requestMetrics, logger, true)  insecureHandlers := New(localCellClient, localMetricCollector, gardenClient, executorClient, evacuatable, requestMetrics, logger, false)  secureHandlers := New(localCellClient, localMetricCollector, gardenClient, executorClient, evacuatable, requestMetrics, logger, true)   for name, handler := range secureHandlers {  insecureHandlers[name] = handler  }  return insecureHandlers }  func logWrap(loggable func(http.ResponseWriter, \*http.Request, lager.Logger), logger lager.Logger) http.HandlerFunc {  return func(w http.ResponseWriter, r \*http.Request) {  requestLog := logger.Session("request", lager.Data{  "method": r.Method,  "request": r.URL.String(),  })   defer requestLog.Debug("done")  requestLog.Debug("serving")   loggable(w, r, requestLog)  } } |

|  |
| --- |
| ○ rep/container\_list\_handler.go package handlers  import (  "code.cloudfoundry.org/executor"  "code.cloudfoundry.org/garden"  GardenClient "code.cloudfoundry.org/garden/client"  "code.cloudfoundry.org/lager"  "encoding/json"  "fmt"  "net/http"  "strconv"  "strings"  "time" ) type ContainerMetricsMetadata struct{   Limits Limits `json:"limits,omitempty"`   UsageMetrics UsageMetrics `json:"usage\_metrics,omitempty"`   Container\_Id string `json:"container\_id,omitempty"`   Interface\_Id string `json:"interface\_id,omitempty"`   Application\_Id string `json:"application\_id,omitempty"`   Application\_Index string `json:"application\_index,omitempty"`   Application\_Name string `json:"application\_name,omitempty"`   Application\_Urls []string `json:"application\_uris,omitempty"`  } type Applications struct{   Limits Limits `json:"limits,omitempty"`   Name string `json:"name,omitempty"`   Application\_Id string `json:"application\_id,omitempty"`   Application\_Version string `json:"application\_version,omitempty"`   Application\_Name string `json:"application\_name,omitempty"`   Application\_Urls []string `json:"application\_uris,omitempty"`   Application\_Index int `json:"application\_index,omitempty"`   Container\_Port uint32 `json:"container\_port,omitempty"`   Space\_Name string `json:"space\_name,omitempty"`   Space\_Id string `json:"space\_id,omitempty"`   Uris []string `json:"uris,omitempty"`    Container\_Id string `json:"container\_id,omitempty"` } type Limits struct {   Fds int32 `json:"fds,omitempty"`   Memory int32 `json:"mem,omitempty"`   Disk int32 `json:"disk,omitempty"`  } type UsageMetrics struct {   MemoryUsageInBytes uint64 `json:"memory\_usage\_in\_bytes"`   DiskUsageInBytes uint64 `json:"disk\_usage\_in\_bytes"`   TimeSpentInCPU time.Duration `json:"time\_spent\_in\_cpu"`  } type ContainerInfo struct {   Container\_Id string `json:"container\_id,omitempty"`   Application\_Id string `json:"application\_id,omitempty"`   Organization\_Id string `json:"organization\_id,omitempty"`   Space\_Id string `json:"space\_id,omitempty"`   Container\_Port uint32 `json:"container\_port,omitempty"`   Interface\_Id string `json:"interface\_id,omitempty"`  } type ContainerListHandler struct {   logger lager.Logger  executorClient executor.Client  gardenClient GardenClient.Client } func NewContainerListHandler(logger lager.Logger, executorClient executor.Client, gardenClient GardenClient.Client) \*ContainerListHandler {   return &ContainerListHandler{   logger: logger,  executorClient: executorClient,  gardenClient: gardenClient,  } } func (c ContainerListHandler) ServeHTTP(w http.ResponseWriter, r \*http.Request, logger lager.Logger) {   var applications []Applications   var containerInfos []ContainerInfo   var containermetrics []ContainerMetricsMetadata   containers, err := c.executorClient.ListContainers(c.logger)  //=============================== Container Metrics ==================================  containerBulkMetrics, err := c.executorClient.GetBulkMetrics(c.logger)   //fmt.Println("======================================================================================================================")  /\*for \_, bulkMetrics := range containerBulkMetrics{  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : key , value :", bulkMetrics)  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : guid :", bulkMetrics.Guid)  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : index :", bulkMetrics.Index)  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : MemoryUsageInBytes :", bulkMetrics.MemoryUsageInBytes)  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : DiskUsageInBytes :", bulkMetrics.DiskUsageInBytes)  fmt.Println("##### container\_list\_handler.go = container Bulk Metrics : TimeSpentInCPU - seconds :", bulkMetrics.TimeSpentInCPU.Seconds())  }\*/  //=====================================================================================  if err != nil {  w.WriteHeader(http.StatusInternalServerError)  c.logger.Error("failed-to-fetch-container", err)  return   }   var app\_host\_port uint32   var app\_index int   for i := range containers {   var application Applications   container := &containers[i]  //fmt.Println("##### container\_list\_handler.go, container :", container.Guid, container)  //fmt.Println("##### container\_list\_handler.go, container Env:", container.Env)  //fmt.Println("##### container\_list\_handler.go, container Network:", container.Network)  //fmt.Println("##### container\_list\_handler.go, container Action:", container.Action)  //fmt.Println("##### container\_list\_handler.go, container Monitor:", container.Monitor) // no  //fmt.Println("##### container\_list\_handler.go, container Setup:", container.Setup) // no  //fmt.Println("##### container\_list\_handler.go, container AllocatedAt:", container.AllocatedAt) // no  //fmt.Println("##### container\_list\_handler.go, container CachedDependencies:", container.CachedDependencies) // no  //fmt.Println("##### container\_list\_handler.go, container EgressRules:", container.EgressRules) // no  //fmt.Println("##### container\_list\_handler.go, container LogConfig:", container.LogConfig) // no  //fmt.Println("##### container\_list\_handler.go, container RunResult:", container.RunResult) // no  //fmt.Println("##### container\_list\_handler.go, container VolumeMounts:", container.VolumeMounts) // no  //fmt.Println("##### container\_list\_handler.go, container Resource:", container.Resource) // no  //fmt.Println("##### container\_list\_handler.go, container RunInfo:", container.RunInfo) // no  //fmt.Println("##### container\_list\_handler.go, container Tags:", container.Tags) // no  //fmt.Println("##### container\_list\_handler.go, container.RootFSPath :", container.RunInfo.RootFSPath) // no  //fmt.Println("##### container\_list\_handler.go, container.Resource.DiskMB :",container.Resource.DiskMB) // no  //fmt.Println("##### container\_list\_handler.go, container.Resource.MemoryMB :",container.Resource.MemoryMB) // no  ////fmt.Println("##### container\_list\_handler.go, container.RunInfo.DiskScope :",container.RunInfo.DiskScope) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.ExternalIP :",container.ExternalIP) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.MemoryMB :",container.MemoryMB) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.State :",container.State) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.Network :",container.Network.Properties) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.ENV :",container.Env) // no  //fmt.Println("##### container\_list\_handler.go, container Ports:", container.Ports) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.Action:", container.RunInfo.Action) // no  //fmt.Println("##### container\_list\_handler.go, container.RunInfo.ActionValue:", container.RunInfo.Action.GetValue()) // no  for \_, value := range container.Ports{   if value.ContainerPort != 2222 {   app\_host\_port = uint32(value.HostPort)  }  }  for \_, value := range container.Env{   if strings.Contains(value.Name, "INSTANCE\_INDEX"){   app\_index, err = strconv.Atoi(value.Value)  }  }  if container.RunInfo.Action.CodependentAction != nil {  action := container.RunInfo.Action.CodependentAction.GetActions()[0].RunAction  if action != nil {   for \_, envs := range action.Env {   if envs.Name == "VCAP\_APPLICATION" {   /\*fmt.Println("##### container\_list\_handler.go, CodependentAction.RunAction.Envs:", envs)\*/  json.Unmarshal([]byte(envs.Value), &application)   //fmt.Println("##### container\_list\_handler.go, Application Info - Id :", application.Application\_Id)  //fmt.Println("##### container\_list\_handler.go, Application Info - Name :", application.Application\_Name)  //fmt.Println("##### container\_list\_handler.go, Application Info - index :", app\_index)  //fmt.Println("##### container\_list\_handler.go, Application Info - Limits :", application.Limits.Disk, application.Limits.Memory)  //fmt.Println("##### container\_list\_handler.go, Application Info - Uris :", application.Uris)   application.Container\_Id = container.Guid   application.Container\_Port = app\_host\_port  application.Application\_Index = app\_index  applications = append(applications, application)  }  }  }  }  }   properties := garden.Properties{}  gardenContainers, err := c.gardenClient.Containers(properties)  var container\_host\_port uint32  for \_, gc := range gardenContainers {   var containerInfo ContainerInfo  gardenContainerInfo, \_ := gc.Info()  /\*fmt.Println("### container\_list\_handler.go - container info : ", gardenContainerInfo)  fmt.Println("### container\_list\_handler.go - container info - container IP: ", gardenContainerInfo.ContainerIP)  fmt.Println("### container\_list\_handler.go - container info - contaienr Path: ", gardenContainerInfo.ContainerPath)  fmt.Println("### container\_list\_handler.go - container info - Host IP: ", gardenContainerInfo.HostIP)  fmt.Println("### container\_list\_handler.go - container info - Properties: ", gardenContainerInfo.Properties)  fmt.Println("### container\_list\_handler.go - container info - ExternalIP: ", gardenContainerInfo.ExternalIP)  fmt.Println("### container\_list\_handler.go - container info - Events: ", gardenContainerInfo.Events)  fmt.Println("### container\_list\_handler.go - container info - MappedPorts: ", gardenContainerInfo.MappedPorts)  fmt.Println("### container\_list\_handler.go - container info - State: ", gardenContainerInfo.State)  fmt.Println("### container\_list\_handler.go - container info - ProcessIDs: ", gardenContainerInfo.ProcessIDs)\*/   var Container\_Ip, External\_Ip string  for key, value := range gardenContainerInfo.Properties {  // fmt.Println("### container\_list\_handler.go - container info - Properties: key - value :", key, value)  // Disable diego CNI  if strings.HasSuffix(key, "host-interface") {  fmt.Println("### container\_list\_handler.go - containerInfo.Interface\_Id :", containerInfo.Interface\_Id)  containerInfo.Interface\_Id = value  break  } else if strings.HasSuffix(key, "container-ip") {  Container\_Ip = value  } else if strings.HasSuffix(key, "external-ip") {  External\_Ip = value  }  }   // Enable diego CNI  if containerInfo.Interface\_Id == "" {  if Container\_Ip != "" && External\_Ip != "" {  containerInfo.Interface\_Id = Container\_Ip + "-" + External\_Ip  } else {  fmt.Println("### container\_list\_handler.go - Not exist Container\_Ip, External\_Ip : ")  }  }   //bugfix container info - MappedPorts: [{61000 8080} {61001 2222} {61002 61001} {61003 61002}]  //fmt.Println("!!!!!!! before container\_list\_handler.go - container info - MappedPorts: ", gardenContainerInfo.MappedPorts)  if len(gardenContainerInfo.MappedPorts) > 0 {  gardenContainerInfo.MappedPorts = gardenContainerInfo.MappedPorts[:len(gardenContainerInfo.MappedPorts)-2]  }  //fmt.Println("!!!!!!! after container\_list\_handler.go - container info - MappedPorts: ", gardenContainerInfo.MappedPorts)  for \_, value := range gardenContainerInfo.MappedPorts{   if value.ContainerPort != 2222 {   container\_host\_port = value.HostPort  }  }  //extract Container ID from gardenContainerInfo.ContainerPath - separator '/' & last value   containerIDPaths := strings.Split(gardenContainerInfo.ContainerPath, "/")  containerInfo.Container\_Id = containerIDPaths[len(containerIDPaths) -1]  for key, props :=range gardenContainerInfo.Properties{  if strings.Contains(key, "app\_id"){   containerInfo.Application\_Id = props  }  }  containerInfo.Container\_Port = container\_host\_port  containerInfos = append(containerInfos, containerInfo)  }  fmt.Println("###### applicationInfos :", applications)  fmt.Println("###### containerInfos :", containerInfos)  for \_, apps :=range applications {   var containermetric ContainerMetricsMetadata   containermetric.Limits = apps.Limits  containermetric.Application\_Id = apps.Application\_Id  containermetric.Application\_Name = apps.Application\_Name  containermetric.Application\_Urls = apps.Application\_Urls  for \_, bulkMetrics :=range containerBulkMetrics {   if apps.Application\_Id == bulkMetrics.Guid && apps.Application\_Index == bulkMetrics.Index {   containermetric.UsageMetrics.MemoryUsageInBytes = bulkMetrics.MemoryUsageInBytes  containermetric.UsageMetrics.DiskUsageInBytes = bulkMetrics.DiskUsageInBytes  containermetric.UsageMetrics.TimeSpentInCPU = bulkMetrics.TimeSpentInCPU  containermetric.Application\_Index = strconv.Itoa(apps.Application\_Index)  }  }  for \_, cons :=range containerInfos {  if apps.Application\_Id == cons.Application\_Id && apps.Container\_Port == cons.Container\_Port {  fmt.Println("!!!!!!! true apps.Application\_Id == cons.Application\_Id && apps.Container\_Port == cons.Container\_Port ")  //containermetric.Container\_Id = cons.Container\_Id  containermetric.Container\_Id = apps.Container\_Id  containermetric.Interface\_Id = cons.Interface\_Id   }  }  containermetrics = append(containermetrics, containermetric)  }  for \_, conmetric :=range containermetrics{  fmt.Println("## container\_list\_handler.go - App & Container Info :", conmetric)  fmt.Println("## container\_list\_handler.go - App & Container Info - Container Id :", conmetric.Container\_Id)  fmt.Println("## container\_list\_handler.go - App & Container Info - Interface Id :", conmetric.Interface\_Id)  fmt.Println("## container\_list\_handler.go - App & Container Info - App Id :", conmetric.Application\_Id)  fmt.Println("## container\_list\_handler.go - App & Container Info - App name :", conmetric.Application\_Name)  fmt.Println("## container\_list\_handler.go - App & Container Info - App uris :", conmetric.Application\_Urls)  }  w.WriteHeader(http.StatusOK)   b, err := json.Marshal(containermetrics)  if err != nil {   c.logger.Error("failed-to-marshalling-containermetrics", err)  }  w.Write(b) } |

|  |
| --- |
| ○ executor/initializer.go package initializer  import (  "bytes"  "crypto/rand"  "crypto/tls"  "crypto/x509"  "encoding/pem"  "errors"  "fmt"  "io/ioutil"  "math"  "os"  "path/filepath"  "sync/atomic"  "time"   "code.cloudfoundry.org/archiver/compressor"  "code.cloudfoundry.org/cacheddownloader"  "code.cloudfoundry.org/clock"  loggingclient "code.cloudfoundry.org/diego-logging-client"  "code.cloudfoundry.org/durationjson"  "code.cloudfoundry.org/executor"  "code.cloudfoundry.org/executor/containermetrics"  "code.cloudfoundry.org/executor/depot"  "code.cloudfoundry.org/executor/depot/containerstore"  "code.cloudfoundry.org/executor/depot/event"  "code.cloudfoundry.org/executor/depot/metrics"  "code.cloudfoundry.org/executor/depot/transformer"  "code.cloudfoundry.org/executor/depot/uploader"  "code.cloudfoundry.org/executor/gardenhealth"  "code.cloudfoundry.org/executor/guidgen"  "code.cloudfoundry.org/executor/initializer/configuration"  "code.cloudfoundry.org/garden"  GardenClient "code.cloudfoundry.org/garden/client"  GardenConnection "code.cloudfoundry.org/garden/client/connection"  "code.cloudfoundry.org/lager"  "code.cloudfoundry.org/systemcerts"  "code.cloudfoundry.org/tlsconfig"  "code.cloudfoundry.org/volman/vollocal"  "code.cloudfoundry.org/workpool"  "github.com/google/shlex"  "github.com/tedsuo/ifrit"  "github.com/tedsuo/ifrit/grouper" )  const (  *PingGardenInterval* = time.Second  *StalledMetricHeartbeatInterval* = 5 \* time.Second  *StalledGardenDuration* = "StalledGardenDuration"  *maxConcurrentUploads* = 5  *metricsReportInterval* = 1 \* time.Minute  *megabytesToBytes* = 1024 \* 1024 )  type executorContainers struct {  gardenClient garden.Client  owner string }  func (containers \*executorContainers) Containers() ([]garden.Container, error) {  return containers.gardenClient.Containers(garden.Properties{  executor.ContainerOwnerProperty: containers.owner,  }) }  //***go:generate*** counterfeiter -o fakes/fake\_cert\_pool\_retriever.go . CertPoolRetriever type CertPoolRetriever interface {  SystemCerts() (\*x509.CertPool, error) }  type systemcertsRetriever struct{}  func (s systemcertsRetriever) SystemCerts() (\*x509.CertPool, error) {  caCertPool, err := systemcerts.SystemCertPool()  if err != nil {  return nil, err  }  if caCertPool == nil {  caCertPool = systemcerts.NewCertPool()  }  return caCertPool.AsX509CertPool(), nil }  type ExecutorConfig struct {  AdvertisePreferenceForInstanceAddress bool `json:"advertise\_preference\_for\_instance\_address"`  AutoDiskOverheadMB int `json:"auto\_disk\_capacity\_overhead\_mb"`  CachePath string `json:"cache\_path,omitempty"`  ContainerInodeLimit uint64 `json:"container\_inode\_limit,omitempty"`  ContainerMaxCpuShares uint64 `json:"container\_max\_cpu\_shares,omitempty"`  ContainerMetricsReportInterval durationjson.Duration `json:"container\_metrics\_report\_interval,omitempty"`  ContainerOwnerName string `json:"container\_owner\_name,omitempty"`  ContainerProxyADSServers []string `json:"container\_proxy\_ads\_addresses,omitempty"`  ContainerProxyConfigPath string `json:"container\_proxy\_config\_path,omitempty"`  ContainerProxyPath string `json:"container\_proxy\_path,omitempty"`  ContainerProxyRequireClientCerts bool `json:"container\_proxy\_require\_and\_verify\_client\_certs"`  ContainerProxyTrustedCACerts []string `json:"container\_proxy\_trusted\_ca\_certs"`  ContainerProxyVerifySubjectAltName []string `json:"container\_proxy\_verify\_subject\_alt\_name"`  ContainerReapInterval durationjson.Duration `json:"container\_reap\_interval,omitempty"`  CreateWorkPoolSize int `json:"create\_work\_pool\_size,omitempty"`  DeclarativeHealthcheckPath string `json:"declarative\_healthcheck\_path,omitempty"`  DeleteWorkPoolSize int `json:"delete\_work\_pool\_size,omitempty"`  DiskMB string `json:"disk\_mb,omitempty"`  EnableContainerProxy bool `json:"enable\_container\_proxy,omitempty"`  EnableDeclarativeHealthcheck bool `json:"enable\_declarative\_healthcheck,omitempty"`  EnableUnproxiedPortMappings bool `json:"enable\_unproxied\_port\_mappings"`  EnvoyConfigRefreshDelay durationjson.Duration `json:"envoy\_config\_refresh\_delay"`  EnvoyConfigReloadDuration durationjson.Duration `json:"envoy\_config\_reload\_duration"`  EnvoyDrainTimeout durationjson.Duration `json:"envoy\_drain\_timeout,omitempty"`  ExportNetworkEnvVars bool `json:"export\_network\_env\_vars,omitempty"` // DEPRECATED. Kept around for dusts compatability  GardenAddr string `json:"garden\_addr,omitempty"`  GardenHealthcheckCommandRetryPause durationjson.Duration `json:"garden\_healthcheck\_command\_retry\_pause,omitempty"`  GardenHealthcheckEmissionInterval durationjson.Duration `json:"garden\_healthcheck\_emission\_interval,omitempty"`  GardenHealthcheckInterval durationjson.Duration `json:"garden\_healthcheck\_interval,omitempty"`  GardenHealthcheckProcessArgs []string `json:"garden\_healthcheck\_process\_args,omitempty"`  GardenHealthcheckProcessDir string `json:"garden\_healthcheck\_process\_dir"`  GardenHealthcheckProcessEnv []string `json:"garden\_healthcheck\_process\_env,omitempty"`  GardenHealthcheckProcessPath string `json:"garden\_healthcheck\_process\_path"`  GardenHealthcheckProcessUser string `json:"garden\_healthcheck\_process\_user"`  GardenHealthcheckTimeout durationjson.Duration `json:"garden\_healthcheck\_timeout,omitempty"`  GardenNetwork string `json:"garden\_network,omitempty"`  GracefulShutdownInterval durationjson.Duration `json:"graceful\_shutdown\_interval,omitempty"`  HealthCheckContainerOwnerName string `json:"healthcheck\_container\_owner\_name,omitempty"`  HealthCheckWorkPoolSize int `json:"healthcheck\_work\_pool\_size,omitempty"`  HealthyMonitoringInterval durationjson.Duration `json:"healthy\_monitoring\_interval,omitempty"`  InstanceIdentityCAPath string `json:"instance\_identity\_ca\_path,omitempty"`  InstanceIdentityCredDir string `json:"instance\_identity\_cred\_dir,omitempty"`  InstanceIdentityPrivateKeyPath string `json:"instance\_identity\_private\_key\_path,omitempty"`  InstanceIdentityValidityPeriod durationjson.Duration `json:"instance\_identity\_validity\_period,omitempty"`  LogRateLimitExceededReportInterval durationjson.Duration `json:"log\_rate\_limit\_exceeded\_report\_interval,omitempty"`  MaxCacheSizeInBytes uint64 `json:"max\_cache\_size\_in\_bytes,omitempty"`  MaxConcurrentDownloads int `json:"max\_concurrent\_downloads,omitempty"`  MaxLogLinesPerSecond int `json:"max\_log\_lines\_per\_second"`  MemoryMB string `json:"memory\_mb,omitempty"`  MetricsWorkPoolSize int `json:"metrics\_work\_pool\_size,omitempty"`  PathToCACertsForDownloads string `json:"path\_to\_ca\_certs\_for\_downloads"`  PathToTLSCACert string `json:"path\_to\_tls\_ca\_cert"`  PathToTLSCert string `json:"path\_to\_tls\_cert"`  PathToTLSKey string `json:"path\_to\_tls\_key"`  PostSetupHook string `json:"post\_setup\_hook"`  PostSetupUser string `json:"post\_setup\_user"`  ProxyMemoryAllocationMB int `json:"proxy\_memory\_allocation\_mb,omitempty"`  ReadWorkPoolSize int `json:"read\_work\_pool\_size,omitempty"`  ReservedExpirationTime durationjson.Duration `json:"reserved\_expiration\_time,omitempty"`  SetCPUWeight bool `json:"set\_cpu\_weight,omitempty"`  SkipCertVerify bool `json:"skip\_cert\_verify,omitempty"`  TempDir string `json:"temp\_dir,omitempty"`  TrustedSystemCertificatesPath string `json:"trusted\_system\_certificates\_path"`  UnhealthyMonitoringInterval durationjson.Duration `json:"unhealthy\_monitoring\_interval,omitempty"`  UseSchedulableDiskSize bool `json:"use\_schedulable\_disk\_size,omitempty"`  VolmanDriverPaths string `json:"volman\_driver\_paths"` }  var (  creationWorkPool, deletionWorkPool \*workpool.WorkPool  metricsWorkPool, readWorkPool \*workpool.WorkPool ) // Adde for PaaS-TA func Initialize(logger lager.Logger, config ExecutorConfig, cellID, zone string,  rootFSes map[string]string, metronClient loggingclient.IngressClient,  //clock clock.Clock) (executor.Client, \*containermetrics.StatsReporter, grouper.Members, error) {  clock clock.Clock) (executor.Client, \*containermetrics.StatsReporter, grouper.Members, GardenClient.Client, error) {   var gardenHealthcheckRootFS string  for \_, rootFSPath := range rootFSes {  gardenHealthcheckRootFS = rootFSPath  break  }   postSetupHook, err := shlex.Split(config.PostSetupHook)  if err != nil {  logger.Error("failed-to-parse-post-setup-hook", err)  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   gardenClient := GardenClient.New(GardenConnection.New(config.GardenNetwork, config.GardenAddr))  err = waitForGarden(logger, gardenClient, metronClient, clock)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }   containersFetcher := &executorContainers{  gardenClient: gardenClient,  owner: config.ContainerOwnerName,  }   creationWorkPool, err = workpool.NewWorkPool(config.CreateWorkPoolSize)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }  deletionWorkPool, err = workpool.NewWorkPool(config.DeleteWorkPoolSize)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }  readWorkPool, err = workpool.NewWorkPool(config.ReadWorkPoolSize)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }  metricsWorkPool, err = workpool.NewWorkPool(config.MetricsWorkPoolSize)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }   err = destroyContainers(gardenClient, containersFetcher, logger)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, nil, err  return nil, nil, nil, nil, err  }   healthCheckWorkPool, err := workpool.NewWorkPool(config.HealthCheckWorkPoolSize)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   certsRetriever := systemcertsRetriever{}  assetTLSConfig, err := TLSConfigFromConfig(logger, certsRetriever, config)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   downloader := cacheddownloader.NewDownloader(10\*time.Minute, int(math.MaxInt8), assetTLSConfig)  uploader := uploader.New(logger, 10\*time.Minute, assetTLSConfig)   cache := cacheddownloader.NewCache(config.CachePath, int64(config.MaxCacheSizeInBytes))  cachedDownloader := cacheddownloader.New(  downloader,  cache,  cacheddownloader.TarTransform,  )   err = cachedDownloader.RecoverState(logger.Session("downloader"))  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   downloadRateLimiter := make(chan struct{}, uint(config.MaxConcurrentDownloads))   transformer := initializeTransformer(  cachedDownloader,  setupWorkDir(logger, config.TempDir),  downloadRateLimiter,  *maxConcurrentUploads*,  uploader,  time.Duration(config.HealthyMonitoringInterval),  time.Duration(config.UnhealthyMonitoringInterval),  time.Duration(config.GracefulShutdownInterval),  healthCheckWorkPool,  clock,  postSetupHook,  config.PostSetupUser,  config.EnableDeclarativeHealthcheck,  gardenHealthcheckRootFS,  config.EnableContainerProxy,  time.Duration(config.EnvoyDrainTimeout),  )   hub := event.NewHub()   totalCapacity, err := fetchCapacity(logger, gardenClient, config)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }  rootFSSizer, err := configuration.GetRootFSSizes(logger, gardenClient, guidgen.DefaultGenerator, config.ContainerOwnerName, rootFSes)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   containerConfig := containerstore.ContainerConfig{  OwnerName: config.ContainerOwnerName,  INodeLimit: config.ContainerInodeLimit,  MaxCPUShares: config.ContainerMaxCpuShares,  SetCPUWeight: config.SetCPUWeight,  ReservedExpirationTime: time.Duration(config.ReservedExpirationTime),  ReapInterval: time.Duration(config.ContainerReapInterval),  MaxLogLinesPerSecond: config.MaxLogLinesPerSecond,  LogRateLimitExceededReportInterval: time.Duration(config.LogRateLimitExceededReportInterval),  }   driverConfig := vollocal.NewDriverConfig()  driverConfig.DriverPaths = filepath.SplitList(config.VolmanDriverPaths)  volmanClient, volmanDriverSyncer := vollocal.NewServer(logger, metronClient, driverConfig)   var proxyConfigHandler containerstore.ProxyManager  if config.EnableContainerProxy {  proxyConfigHandler = containerstore.NewProxyConfigHandler(  logger,  config.ContainerProxyPath,  config.ContainerProxyConfigPath,  config.ContainerProxyTrustedCACerts,  config.ContainerProxyVerifySubjectAltName,  config.ContainerProxyRequireClientCerts,  time.Duration(config.EnvoyConfigReloadDuration),  clock,  config.ContainerProxyADSServers,  )  } else {  proxyConfigHandler = containerstore.NewNoopProxyConfigHandler()  }   instanceIdentityHandler := containerstore.NewInstanceIdentityHandler(  config.InstanceIdentityCredDir,  "/etc/cf-instance-credentials",  )   credManager, err := CredManagerFromConfig(logger, metronClient, config, clock, proxyConfigHandler, instanceIdentityHandler)  if err != nil {  // Adde for PaaS-TA  //return nil, nil, grouper.Members{}, err  return nil, nil, grouper.Members{}, nil, err  }   containerStore := containerstore.New(  containerConfig,  &totalCapacity,  gardenClient,  containerstore.NewDependencyManager(cachedDownloader, downloadRateLimiter),  volmanClient,  credManager,  clock,  hub,  transformer,  config.TrustedSystemCertificatesPath,  metronClient,  rootFSSizer,  config.EnableDeclarativeHealthcheck,  config.DeclarativeHealthcheckPath,  proxyConfigHandler,  cellID,  config.EnableUnproxiedPortMappings,  config.AdvertisePreferenceForInstanceAddress,  )   depotClient := depot.NewClient(  totalCapacity,  containerStore,  gardenClient,  volmanClient,  hub,  creationWorkPool,  deletionWorkPool,  readWorkPool,  metricsWorkPool,  )   healthcheckSpec := garden.ProcessSpec{  Path: config.GardenHealthcheckProcessPath,  Args: config.GardenHealthcheckProcessArgs,  User: config.GardenHealthcheckProcessUser,  Env: config.GardenHealthcheckProcessEnv,  Dir: config.GardenHealthcheckProcessDir,  }   gardenHealthcheck := gardenhealth.NewChecker(  gardenHealthcheckRootFS,  config.HealthCheckContainerOwnerName,  time.Duration(config.GardenHealthcheckCommandRetryPause),  healthcheckSpec,  gardenClient,  guidgen.DefaultGenerator,  )   metricsCache := &atomic.Value{}  containerStatsReporter := containermetrics.NewStatsReporter(  metronClient,  config.EnableContainerProxy,  float64(config.ProxyMemoryAllocationMB\*megabytesToBytes),  metricsCache,  )  cpuSpikeReporter := containermetrics.NewCPUSpikeReporter(metronClient)   reportersRunner := containermetrics.NewReportersRunner(  logger,  time.Duration(config.ContainerMetricsReportInterval),  clock,  depotClient,  containerStatsReporter,  cpuSpikeReporter,  )   return depotClient, containerStatsReporter,  grouper.Members{  {"volman-driver-syncer", volmanDriverSyncer},  {"metrics-reporter", &metrics.Reporter{  ExecutorSource: depotClient,  Interval: *metricsReportInterval*,  Clock: clock,  Logger: logger,  MetronClient: metronClient,  Tags: map[string]string{"zone": zone},  }},  {"hub-closer", closeHub(logger, hub)},  {"container-metrics-reporter", reportersRunner},  {"garden\_health\_checker", gardenhealth.NewRunner(  time.Duration(config.GardenHealthcheckInterval),  time.Duration(config.GardenHealthcheckEmissionInterval),  time.Duration(config.GardenHealthcheckTimeout),  logger,  gardenHealthcheck,  depotClient,  metronClient,  clock,  )},  {"registry-pruner", containerStore.NewRegistryPruner(logger)},  {"container-reaper", containerStore.NewContainerReaper(logger)},  },  // Adde for PaaS-TA  gardenClient,  nil }  // Until we get a successful response from garden, // periodically emit metrics saying how long we've been trying // while retrying the connection indefinitely. func waitForGarden(logger lager.Logger, gardenClient GardenClient.Client, metronClient loggingclient.IngressClient, clock clock.Clock) error {  pingStart := clock.Now()  logger = logger.Session("wait-for-garden", lager.Data{"initialTime:": pingStart})  pingRequest := clock.NewTimer(0)  pingResponse := make(chan error)  heartbeatTimer := clock.NewTimer(*StalledMetricHeartbeatInterval*)   for {  select {  case <-pingRequest.C():  go func() {  logger.Info("ping-garden", lager.Data{"wait-time-ns:": clock.Since(pingStart)})  pingResponse <- gardenClient.Ping()  }()   case err := <-pingResponse:  switch err.(type) {  case nil:  logger.Info("ping-garden-success", lager.Data{"wait-time-ns:": clock.Since(pingStart)})  // send 0 to indicate ping responded successfully  sendError := metronClient.SendDuration(*StalledGardenDuration*, 0)  if sendError != nil {  logger.Error("failed-to-send-stalled-duration-metric", sendError)  }  return nil  case garden.UnrecoverableError:  logger.Error("failed-to-ping-garden-with-unrecoverable-error", err)  return err  default:  logger.Error("failed-to-ping-garden", err)  pingRequest.Reset(*PingGardenInterval*)  }   case <-heartbeatTimer.C():  logger.Info("emitting-stalled-garden-heartbeat", lager.Data{"wait-time-ns:": clock.Since(pingStart)})  sendError := metronClient.SendDuration(*StalledGardenDuration*, clock.Since(pingStart))  if sendError != nil {  logger.Error("failed-to-send-stalled-duration-heartbeat-metric", sendError)  }   heartbeatTimer.Reset(*StalledMetricHeartbeatInterval*)  }  } }  func fetchCapacity(logger lager.Logger, gardenClient GardenClient.Client, config ExecutorConfig) (executor.ExecutorResources, error) {  capacity, err := configuration.ConfigureCapacity(gardenClient, config.MemoryMB, config.DiskMB, config.MaxCacheSizeInBytes, config.AutoDiskOverheadMB, config.UseSchedulableDiskSize)  if err != nil {  logger.Error("failed-to-configure-capacity", err)  return executor.ExecutorResources{}, err  }   logger.Info("initial-capacity", lager.Data{  "capacity": capacity,  })   return capacity, nil }  func destroyContainers(gardenClient garden.Client, containersFetcher \*executorContainers, logger lager.Logger) error {  logger.Info("executor-fetching-containers-to-destroy")  containers, err := containersFetcher.Containers()  if err != nil {  logger.Error("executor-failed-to-get-containers", err)  return err  }   logger.Info("executor-fetched-containers-to-destroy", lager.Data{"num-containers": len(containers)})   type containerDeletionResult struct {  handle string  err error  }   errInfoChannel := make(chan containerDeletionResult, len(containers))  for \_, container := range containers {  go func(c garden.Container) {  deletionWorkPool.Submit(func() {  err := gardenClient.Destroy(c.Handle())  errInfoChannel <- containerDeletionResult{handle: c.Handle(), err: err}  })  }(container)  }   for \_, \_ = range containers {  select {  case result := <-errInfoChannel:  if result.err != nil {  logger.Error("executor-failed-to-destroy-container", result.err, lager.Data{  "handle": result.handle,  })  return result.err  } else {  logger.Info("executor-destroyed-stray-container", lager.Data{  "handle": result.handle,  })  }  }  }   return nil }  func setupWorkDir(logger lager.Logger, tempDir string) string {  workDir := filepath.Join(tempDir, "executor-work")   err := os.RemoveAll(workDir)  if err != nil {  logger.Error("working-dir.cleanup-failed", err)  os.Exit(1)  }   err = os.MkdirAll(workDir, 0755)  if err != nil {  logger.Error("working-dir.create-failed", err)  os.Exit(1)  }   return workDir }  func initializeTransformer(  cache cacheddownloader.CachedDownloader,  workDir string,  downloadRateLimiter chan struct{},  maxConcurrentUploads uint,  uploader uploader.Uploader,  healthyMonitoringInterval time.Duration,  unhealthyMonitoringInterval time.Duration,  gracefulShutdownInterval time.Duration,  healthCheckWorkPool \*workpool.WorkPool,  clock clock.Clock,  postSetupHook []string,  postSetupUser string,  useDeclarativeHealthCheck bool,  declarativeHealthcheckRootFS string,  enableContainerProxy bool,  drainWait time.Duration, ) transformer.Transformer {  var options []transformer.Option  compressor := compressor.NewTgz()   options = append(options, transformer.WithSidecarRootfs(declarativeHealthcheckRootFS))   if useDeclarativeHealthCheck {  options = append(options, transformer.WithDeclarativeHealthchecks())  }   if enableContainerProxy {  options = append(options, transformer.WithContainerProxy(drainWait))  }   options = append(options, transformer.WithPostSetupHook(postSetupUser, postSetupHook))   return transformer.NewTransformer(  clock,  cache,  uploader,  compressor,  downloadRateLimiter,  make(chan struct{}, maxConcurrentUploads),  workDir,  healthyMonitoringInterval,  unhealthyMonitoringInterval,  gracefulShutdownInterval,  healthCheckWorkPool,  options...,  ) }  func closeHub(logger lager.Logger, hub event.Hub) ifrit.Runner {  return ifrit.RunFunc(func(signals <-chan os.Signal, ready chan<- struct{}) error {  close(ready)  signal := <-signals  hub.Close()  hubLogger := logger.Session("close-hub")  hubLogger.Info("signalled", lager.Data{"signal": signal.String()})  return nil  }) }  func TLSConfigFromConfig(logger lager.Logger, certsRetriever CertPoolRetriever, config ExecutorConfig) (\*tls.Config, error) {  var tlsConfig \*tls.Config  var err error   caCertPool, err := certsRetriever.SystemCerts()  if err != nil {  return nil, err  }  if (config.PathToTLSKey != "" && config.PathToTLSCert == "") || (config.PathToTLSKey == "" && config.PathToTLSCert != "") {  return nil, errors.New("The TLS certificate or key is missing")  }   if config.PathToTLSCACert != "" {  caCertPool, err = appendCACerts(caCertPool, config.PathToTLSCACert)  if err != nil {  return nil, err  }  }   if config.PathToCACertsForDownloads != "" {  caCertPool, err = appendCACerts(caCertPool, config.PathToCACertsForDownloads)  if err != nil {  return nil, err  }  }   if config.PathToTLSKey != "" && config.PathToTLSCert != "" {  tlsConfig, err = tlsconfig.Build(  tlsconfig.WithInternalServiceDefaults(),  tlsconfig.WithIdentityFromFile(config.PathToTLSCert, config.PathToTLSKey),  ).Client(  tlsconfig.WithAuthority(caCertPool),  )  if err != nil {  logger.Error("failed-to-configure-tls", err)  return nil, err  }  tlsConfig.InsecureSkipVerify = config.SkipCertVerify  // Make the cipher suites less restrictive as we cannot control what cipher  // suites asset servers support  tlsConfig.CipherSuites = nil  } else {  tlsConfig = &tls.Config{  RootCAs: caCertPool,  InsecureSkipVerify: config.SkipCertVerify,  MinVersion: tls.VersionTLS12,  }  }   return tlsConfig, nil }  func CredManagerFromConfig(logger lager.Logger, metronClient loggingclient.IngressClient, config ExecutorConfig, clock clock.Clock, handlers ...containerstore.CredentialHandler) (containerstore.CredManager, error) {  if config.InstanceIdentityCredDir != "" {  logger.Info("instance-identity-enabled")  keyData, err := ioutil.ReadFile(config.InstanceIdentityPrivateKeyPath)  if err != nil {  return nil, err  }  keyBlock, \_ := pem.Decode(keyData)  if keyBlock == nil {  return nil, errors.New("instance ID key is not PEM-encoded")  }  privateKey, err := x509.ParsePKCS1PrivateKey(keyBlock.Bytes)  if err != nil {  return nil, err  }   certData, err := ioutil.ReadFile(config.InstanceIdentityCAPath)  if err != nil {  return nil, err  }  certBlock, \_ := pem.Decode(certData)  if certBlock == nil {  return nil, errors.New("instance ID CA is not PEM-encoded")  }  certs, err := x509.ParseCertificates(certBlock.Bytes)  if err != nil {  return nil, err  }   if config.InstanceIdentityValidityPeriod <= 0 {  return nil, errors.New("instance ID validity period needs to be set and positive")  }   return containerstore.NewCredManager(  logger,  metronClient,  time.Duration(config.InstanceIdentityValidityPeriod),  rand.Reader,  clock,  certs[0],  privateKey,  handlers...,  ), nil  }   logger.Info("instance-identity-disabled")  return containerstore.NewNoopCredManager(), nil }  func (config \*ExecutorConfig) Validate(logger lager.Logger) bool {  valid := true   if config.ContainerMaxCpuShares == 0 {  logger.Error("max-cpu-shares-invalid", nil)  valid = false  }   if config.HealthyMonitoringInterval <= 0 {  logger.Error("healthy-monitoring-interval-invalid", nil)  valid = false  }   if config.UnhealthyMonitoringInterval <= 0 {  logger.Error("unhealthy-monitoring-interval-invalid", nil)  valid = false  }   if config.GardenHealthcheckInterval <= 0 {  logger.Error("garden-healthcheck-interval-invalid", nil)  valid = false  }   if config.GardenHealthcheckProcessUser == "" {  logger.Error("garden-healthcheck-process-user-invalid", nil)  valid = false  }   if config.GardenHealthcheckProcessPath == "" {  logger.Error("garden-healthcheck-process-path-invalid", nil)  valid = false  }   if config.PostSetupHook != "" && config.PostSetupUser == "" {  logger.Error("post-setup-hook-requires-a-user", nil)  valid = false  }   return valid }  func appendCACerts(caCertPool \*x509.CertPool, pathToCA string) (\*x509.CertPool, error) {  certBytes, err := ioutil.ReadFile(pathToCA)  if err != nil {  return nil, fmt.Errorf("Unable to open CA cert bundle '%s'", pathToCA)  }   certBytes = bytes.TrimSpace(certBytes)   if len(certBytes) > 0 {  if ok := caCertPool.AppendCertsFromPEM(certBytes); !ok {  return nil, errors.New("unable to load CA certificate")  }  }   return caCertPool, nil } |

1. 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.) [↑](#footnote-ref-1)