Руководство по установке

Компонента Request Validator (REQV)

Продукта Platform V Synapse Service Mesh (SSM)

ОГЛАВЛЕНИЕ

Руководство по установке

Термины и определения

Термин/аббревиатура	Определение
API	Application Programming Interface, программный интерфейс приложения
URL	Uniform Resource Locator, унифицированный адрес ресурса
Platform V Synapse Service Mesh / SSM	Программный продукт на базе Istio SE, обеспечивающий возможность создания сервисной сети поверх Платформенной в Kubernetes
Istio SE	Настраиваемая сервисная сетка с открытым исходным кодом, служащая для взаимодействия, мониторинга и обеспечения безопасности контейнеров в кластере Kubernetes
Контрольная панель	Проект, где запущены управляющие приложения SSM
Платформа	Платформа оркестрации приложений с средствами автоматизации и управления на основе политик, например Kubernetes
Граничный прокси/ IGEG	Компонент Граничный прокси продукта Platform V Synapse Service Mesh
Request Validator/ REQV	Компонент Request Validator продукта Platform V Synapse Service Mesh

Системные требования

Для функционирования компоненты Request Validator продукта Platform V Synapse Service Mesh (далее - Сервис валидации) предъявляются требования по наличию следующего программного обеспечения:

Наименование ПО	Версия ПО	Назначение ПО
Kubernetes	1.19 и выше	Платформа оркестрации приложений с средствами автоматизации и управления на основе политик

Перед установкой проверьте соблюдение следующих условий:

1. При работе компоненты REQV с IGEG требуется наличие контрольной панели SSM;
2. В кластере создан проект в котором будет разворачиваться валидатор запросов. При работе компоненты REQV с IGEG требуется подключить проект к контрольной панели SSM;
3. В проекте создана учетная запись с правами на загрузку артефактов (администратор проекта);
4. Получена ссылка на целевой Docker-репозиторий;
5. В проект добавлен секрет для загрузки docker-образов из целевого Docker-репозитория;
6. В проекте имеются свободные ресурсы по лимитам и реквестам не менее, чем зарезервировано в конфигурационных артефактах;
7. При установке с использованием консоли, на рабочем месте должен быть установлен клиент Kubernetes (kubectl).

Для работы компонента REQV рекомендуется развернуть следующие компоненты продукта Platform V: | Наименование компонента Platform V | Код компонента Platform V | Назначение компонента Platform V | | --- | --- | --- | | Граничный прокси | IGEG | Граничный прокси из состава продукта Platform V Synapse Service Mesh, предназначен для обеспечения управляемого вызова интеграционных сервисов прикладной части в проекте Kubernetes. Компонент REQV добавляется в один деплоймент к IGEG | | TC Аудит | AUDT | Компонент продукта Platform V Audit, предназначенный для хранения логов аудита. Не является обязательным для работы REQV, так как компонент REQV может осуществлять отправку событий аудита в любое локальное хранилище логов. Рекомендуем использовать AUDT |

Минимальные ресурсы

Минимальные ресурсы, необходимые для компонента:

Контейнер	Режим работы приложения	CPU Request	Memory Request	CPU Limit	Memory Limit
istio-proxy	Ingress Gateway	300	300	300	300

Контейнер	Режим работы приложения	CPU Request	Memory Request	CPU Limit	Memory Limit
istio-proxy	Egress Gateway	300	300	300	300
Прикладное приложение	Отдельный Pod	300	300	300	300

Количество задействованных реплик компонент IGEG и прикладных приложений зависит от объема принимаемых данных.

Требования к КТС

Требования к техническим компонентам

Требования к КТС — это размер ресурсов CPU, памяти, выделяемых в Kubernetes. Их значение вычисляется для конкретного разворачиваемого инстанса в зависимости от профиля нагрузки.

Принципы размещения сервиса на КТС

Компонент REQV разворачивается в виде контейнера в деплойменте прикладного приложения или в деплойменте IGEG.

Специфические технологические решения

Отсутствуют.

Установка

Установка сервиса валидации

В зависимости от целей бизнес сервиса, сервис валидации может быть установлен в следующих режимах:

- Работа в режиме сайдкар в Ingress Gateway Deployment
- Работа в режима сайдкар в Egress Gateway Deployment
- Работа в режиме отдельного сервиса Валидации

Состав дистрибутива

Дистрибутив содержит файлы:

- ConfigMap.yml (validator-config)
- EnvoyFilter-ingress.yml (validator-ingress)
- Configmap-schemas.yml (validator-schemas)
- Deployment.yml (ingressgateway)

- Gateway.yml (ingressgateway-gw)
- Service-ingress.yml (ingressgateway)
- VirtualService.yml (validator-ingress)

1. Конфигурация приложения (с примерами настройки валидации)

validator-config

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-config
  labels:
    app: validator
data:
  app.yaml: |
    log_level: info
    port: 8787
    schemas:
      - name: foo
        method: POST
        path: '/foo'
        schema: foo_schema
        type: 'json'
        error_code: 401
        error_response: '{"x-request-id": "{{get .Header "x-request-id"}}", "error": "{{get
.Error "error_message"}}"}'
        error_headers:
          - key: Content-Type
            value: application/json
          - key: foo
            value: '{{get .Header "foo"}}'
        headers:
          - key: x-foo-first
            value: foo1
          - key: x-foo-second
            value: foo2
      - name: bar
        method: POST
        path: '/bar'
        schema: bar_schema
        type: 'xml'
    audit:
      enable: true
      url: audit2-client.ca.sbrf.ru
      metamodel:
        module: 'validator-audit-module'
      eventmodel:
        module: 'validator-audit-module'
        user_node: ''
        user_login: ''
        user_name: ''
        session: ''
        node_id: 'project.name'
```

1. Конфигурация со схемами валидации

validator-schemas

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-schemas
```

```
  labels:
    app: validator
data:
  foo.json: |
    {
      "$id": "http://json-schema.org/draft/2019-09/json-schema-core.html",
      "$schema": "https://json-schema.org/draft/2019-09/schema",
      "$comment": "sample comment",
      "title": "Config dump",
      "type": "object",
      "properties": {
        "configs": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "@type": {
                "type": "string"
              },
              "last_updated": {
                "type": "string"
              },
              "bootstrap": {
                "type": "object",
                "properties": {
                  "admin": {
                    "type": "object"
                  },
                  "dynamic_resources": {
                    "type": "object"
                  },
                  "node": {
                    "type": "object"
                  },
                  "static_resources": {
                    "type": "object"
                  },
                  "stats_config": {
                    "type": "object"
                  }
                },
                "required": ["admin", "dynamic_resources", "node", "static_resources",
"stats_config"]
              }
            },
            "required": ["@type", "last_updated", "bootstrap"]
          }
        }
      },
      "required": [
        "configs"
      ]
    }
  bar.xsd: |
    <?xml version="1.0" encoding="UTF-8" ?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="address">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="street" type="xs:string"/>
                    <xs:element name="street-number" type="xs:string"/>
                    <xs:element name="city" type="xs:string"/>
                    <xs:element name="zip" type="xs:string"/>
                    <xs:element name="country" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
```

```
        </xs:element>
    </xs:schema>
```

1. Конфигурация EnvoyFilter для Ingress Gateway (пример)

validator-ingress

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: validator-ingress
spec:
  workloadSelector:
    labels:
      app: ingress-${PROJECT_NAME}
      istio: ingress-${PROJECT_NAME}
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: envoy.http_connection_manager
          portNumber: 8080
      patch:
        operation: INSERT_BEFORE
        value:
          config:
            failure_mode_allow: false
            grpc_service:
              google_grpc:
                stat_prefix: ext_authz
                target_uri: '127.0.0.1:50051'
              timeout: 2s
            with_request_body:
              allow_partial_message: true
              max_request_bytes: 16777216
          name: envoy.ext_authz
```

1. Пример EnvoyFilter для Egress Gateway

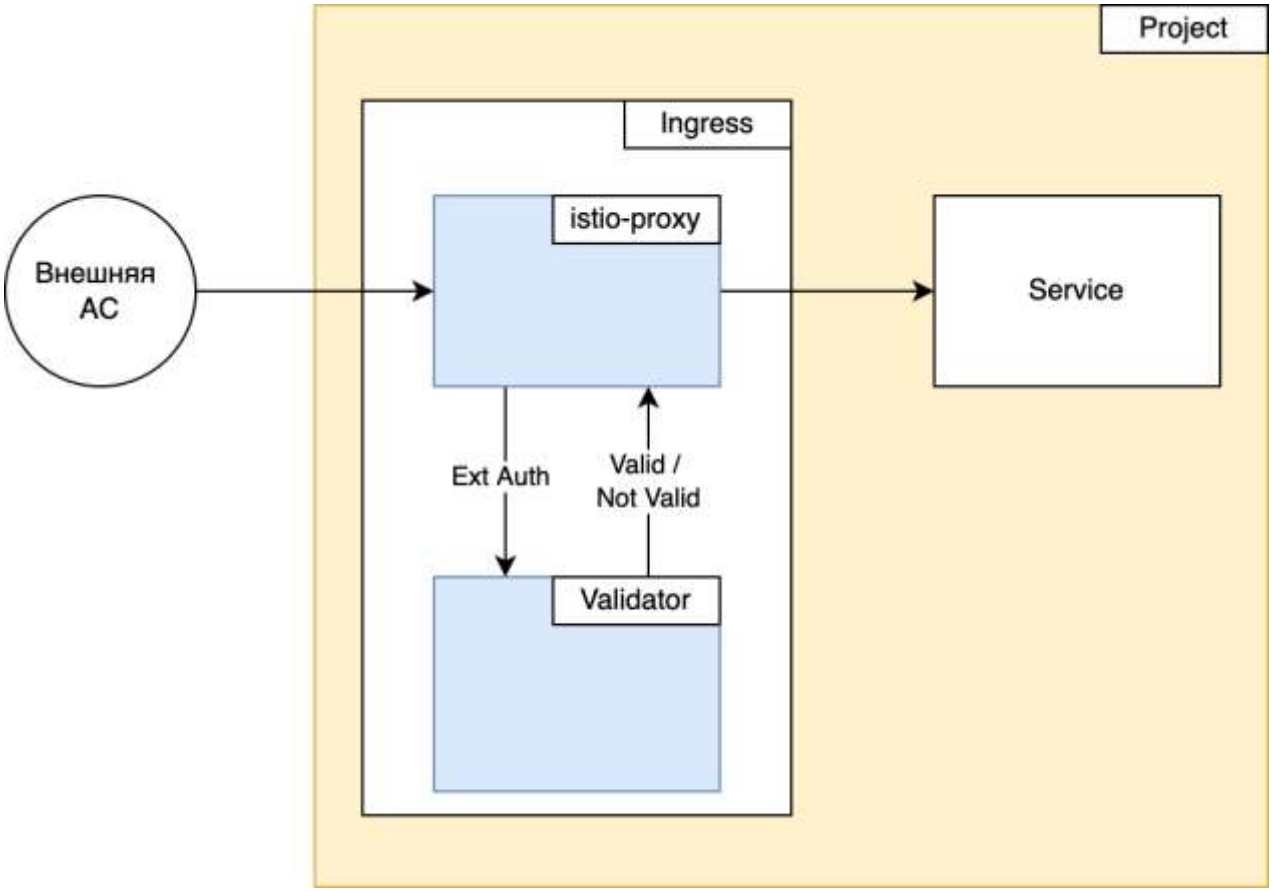
validator-egress

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: validator-egress
spec:
  workloadSelector:
    labels:
      app: {EGRESS_MATCH_LABEL} #заменить на лейбл Egress
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: envoy.http_connection_manager
              subFilter:
                name: envoy.router
          portNumber: 8088
      patch:
```

```
        operation: INSERT_BEFORE
        value:
          name: envoy.lua
          typed_config:
            '@type': type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua
            inlineCode: |
              local path = ""
              local method = ""
              function envoy_on_response(response_handle)
                local authentication_request = {
                  [":method"] = "POST",
                  [":path"] = "/validate",
                  [":authority"] = "auth.local",
                  ["path"] = path,
                  ["method"] = method
                }

                local body = response_handle:body()

                if body == nil or body == '' then
                  body = "0"
                end

                jsonBody = tostring(body:getBytes(0, body:length()))

                local response_headers, response_body =
response_handle:httpCall("outbound|8787||auth.local", authentication_request, jsonBody, 100)
                if response_headers[":status"] ~= "200" then
                  response_handle:headers():replace(":status", "403")
                end
              end
              function envoy_on_request(request_handle)
                path = request_handle:headers():get(":path")
                method = request_handle:headers():get(":method")
              end
```

Установка схемы валидации входящих запросов на Ingress Gateway

Запрос приходит в сервис Ingress и согласно конфигурации EnvoyFilter отправляется на валидацию в контейнер с Валидатором. После проверки тела запроса приходит ответ с результатом валидации. При неуспешной валидации запроса отправляется ошибка вызывающей системе. При успешной валидации запроса, отправляется в прикладной сервис проекта (согласно конфигурации VirtualService).



Шаги:

1. Добавление Ingress Gateway Deployment

Взять из дистрибутива, подставив имя неймспейса контрольной панели с параметрами ISTIO_CONTROL_PLANE и ISTIO_PROXY_IMAGE. Также можно запросить у администраторов SSM конфигурацию Ingress Gateway Deployment для установки в проект или выгрузить из поставки для компонента POLM. Или

1. Добавление контейнера с сервисом валидации

Сервис валидации устанавливается в Ingress Gateway Deployment в раздел containers.

{REQV_IMAGE} - подставить актуальную версию docker образа сервиса валидации.

Ingress Deployment: Containers

```
containers:
- name: validator
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 300m
      memory: 300Mi
  terminationMessagePath: /dev/termination-log
  env:
    - name: PROJECT_NAME
      valueFrom:
```

```
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
  ports:
    - containerPort: 50051
      protocol: TCP
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - name: validator-schemas
      readOnly: true
      mountPath: /schemas
    - name: validator-config
      mountPath: /config
  terminationMessagePolicy: File
  image: >-
    {REQV_IMAGE}
```

Добавление конфигураций сервиса валидатор из ConfigMap

Ingress Deployment: Volumes

```
volumes:
  - name: validator-config
    configMap:
      name: validator-config
      defaultMode: 420
  - name: validator-schemas
    configMap:
      name: validator-schemas
      defaultMode: 420
```

1. Добавление Gateway для Ingress Gateway

Конфигурацию Gateway необходимо привязать к Ingress Gateway посредством лейблов selector. Также, указывается host

Gateway

```
kind: Gateway
apiVersion: networking.istio.io/v1alpha3
metadata:
  name: ingressgateway-gw
  labels:
    app: validator
    template: validator
spec:
  servers:
    - hosts:
        - ingress-json-synapse.ru
      port:
        name: http
        number: 8080
        protocol: HTTP
  selector:
    app: ingressgateway-sandbox
    istio: ingressgateway-sandbox
```

1. Добавление VirtualService

Пример конфигурации VirtualService для перенаправления валидного запроса от Ingress Gateway к бизнес-сервису

VirtualService

```
kind: VirtualService
apiVersion: networking.istio.io/v1alpha3
metadata:
  name: validator-ingress
  labels:
    app: validator
    template: validator
spec:
  hosts:
    - ingress-json-sandbox.ru
  gateways:
    - ingressgateway-gw
  http:
    - match:
        - uri:
            prefix: /
      route:
        - destination:
            host: ping-app-svc
            port:
              number: 8787
  exportTo:
```

1. Добавление EnvoyFilter

Пример EnvoyFilter для перенаправления запроса от Ingress Gateway к сайдкар сервису валидатора

EnvoyFilter

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: validator-ingress
spec:
    workloadSelector:
    labels:
      app: istio-ingressgateway
      istio: ingressgateway
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: envoy.filters.network.http_connection_manager
          portNumber: 8282
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.filters.http.ext_authz
          typed_config:
            '@type': >-
              type.googleapis.com/envoy.extensions.filters.http.ext_authz.v3.ExtAuthz
            transport_api_version: V3
            grpc_service:
              google_grpc:
                stat_prefix: ext_authz
                target_uri: 127.0.0.1:50052
              timeout: 2s
            status_on_error:
```

```
            code: 503
          with_request_body:
            allow_partial_message: true
            max_request_bytes: 16777216
            pack_as_bytes: true
```

1. Добавление конфигурации приложения (пример)

ConfigMap: App

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-config
  labels:
    app: validator
data:
  app.yaml: |
    port: 8787
    log_level: info
    schemas:
      - name: 'foo'
        method: POST
        path: '/ping'
        schema: /schemas/foo.json
        type: 'json'
        error_response: '{"x-request-id": "{{get .Header "x-request-id"}}", "error": "{{get
.Error "error_message"}}"}'
        error_headers:
          - key: foo-bar-x
            value: foo.bar
          - key: foo
            value: '{{get .Header "foo"}}'
      - name: bar
        method: POST
        path: '/'
        schema: /schemas/bar.xsd
        type: 'xml'
        error_code: 401
        headers:
          - key: x-foo-first
            value: foo1
      - name: pass-path
        method: GET
        path: '/json'
        schema: ''
        type: 'pass'
```

1. Добавление конфигурации схем (пример)

ConfigMap

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: json-validator
  labels:
    app: ${APP_NAME}
data:
  foo.json: |
    {
      "$id": "http://json-schema.org/draft/2019-09/json-schema-core.html",
      "$schema": "https://json-schema.org/draft/2019-09/schema",
      "$comment": "sample comment",
```

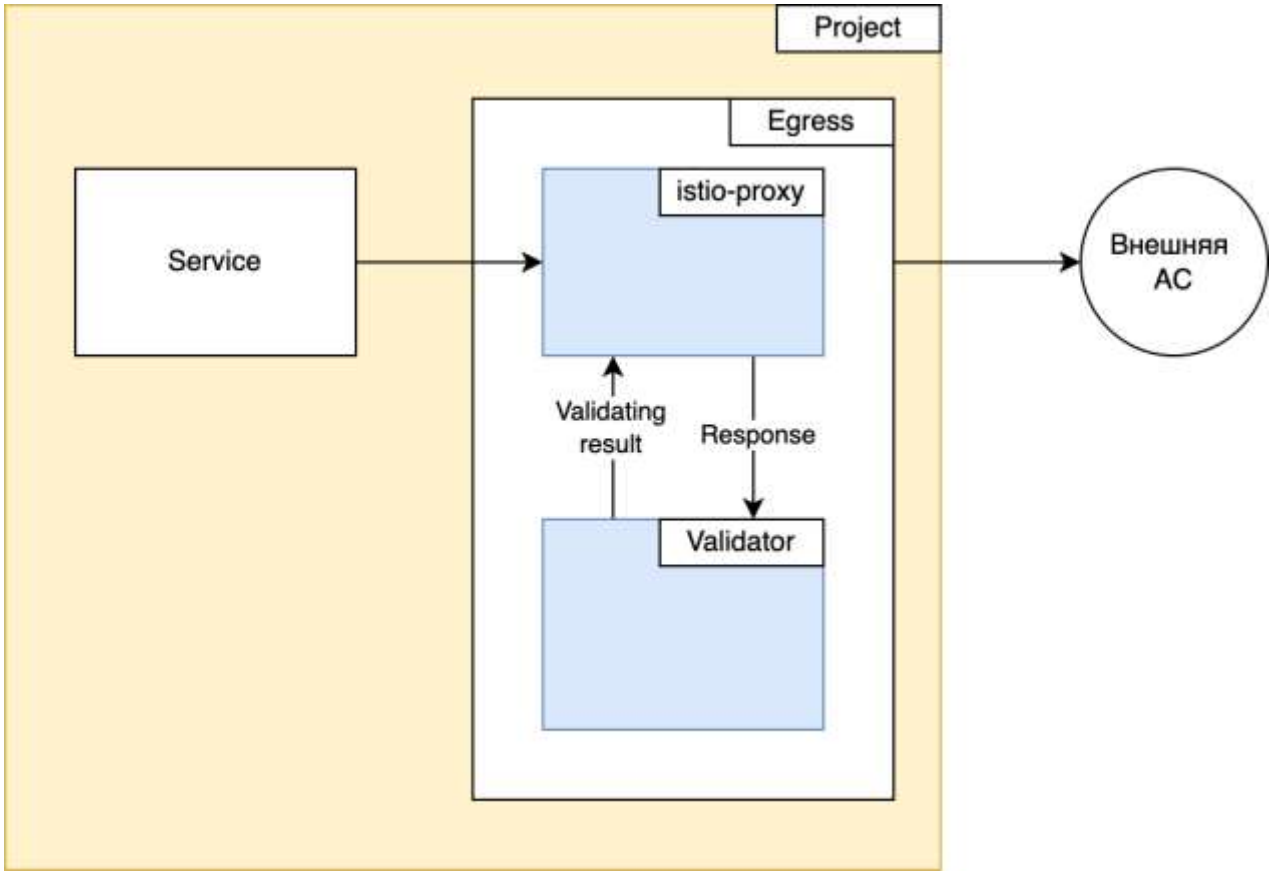
```
    "title": "Config dump",
    "type": "object",
    "properties": {
      "configs": {
        "type": "array",
        "items": {
          "type": "object",
          ...
  }
bar.xsd: |
  <?xml version="1.0" encoding="UTF-8" ?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="address">
          <xs:complexType>
              <xs:sequence>
                  <xs:element name="street" type="xs:string"/>
                  <xs:element name="street-number" type="xs:string"/>
                  <xs:element name="city" type="xs:string"/>
                  <xs:element name="zip" type="xs:string"/>
                  <xs:element name="country" type="xs:string"/>
              </xs:sequence>
          </xs:complexType>
      </xs:element>
  </xs:schema>
```

Схема валидации входящих запросов на Egress Gateway

Запрос из прикладного сервиса во внешнюю АС проходит через Egress Gateway. Ответ от АС отправляется на Валидатор.

При успешном прохождении валидации - Ответ отправляется в прикладной сервис с HTTP кодом "200".

При неуспешной валидации - На прикладной сервис приходит ответ с кодом "403" (либо согласно заданными настройкам в EnvoyFilter Egress Gateway).



1. Добавление Deployment для Egress Gateway

Запросить у администратора контрольной панели SSM конфигурацию Deployment для Egress Gateway для установки в проект или выгрузить из поставки для компонента POLM.

1. Добавление контейнера с сервисом валидации

Сервис валидации устанавливается в конфигурацию Deployment для Egress Gateway в раздел containers.

{REGISTRY_LINK} - подставить актуальную версию docker образа сервиса валидации.

Egress Deployment: Containers

```
containers:
- name: validator
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 300m
      memory: 300Mi
  terminationMessagePath: /dev/termination-log
  env:
    - name: PROJECT_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
  ports:
```

```
    - containerPort: 50051
      protocol: TCP
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - name: validator-schemas
      readOnly: true
      mountPath: /schemas
    - name: validator-config
      mountPath: /config
  terminationMessagePolicy: File
  image: >-
    {REGISTRY_LINK}
```

1. Добавление Gateway для Egress Gateway

Конфигурацию Gateway необходимо привязать к Egress Gateway посредством лейблов selector. Также указывается корректный host

Gateway

```
kind: Gateway
apiVersion: networking.istio.io/v1alpha3
metadata:
  name: ingressgateway-gw
  labels:
    app: validator
    template: validator
spec:
  servers:
    - hosts:
        - ingress-json-sandbox.ru
      port:
        name: http
        number: 8080
        protocol: HTTP
  selector:
    app: ingressgateway-sandbox
    istio: ingressgateway-sandbox
```

1. Добавление ServiceEntry

Пример конфигурации ServiceEntry для перенаправления запроса в контейнер валидации исходящего запроса

VirtualService

```
kind: ServiceEntry
apiVersion: networking.istio.io/v1alpha3
metadata:
  name: auth-local
spec:
  hosts:
    - auth.local
  ports:
    - name: http
      number: 8787
      protocol: HTTP
  location: MESH_INTERNAL
  resolution: STATIC
  endpoints:
    - address: 127.0.0.1
```

1. Добавление EnvoyFilter

Пример EnvoyFilter для перенаправления запроса от Ingress Gateway к сайдкар сервису валидатора

Egress EnvoyFilter

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: validator-egress
spec:
  workloadSelector:
    labels:
      app: {EGRESS_MATCH_LABEL} #заменить на лейбл Egress
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: envoy.http_connection_manager
              subFilter:
                name: envoy.router
          portNumber: 8088
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.lua
          typed_config:
            '@type': type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua
            inlineCode: |
              local path = ""
              local method = ""
              function envoy_on_response(response_handle)
                local authentication_request = {
                  [":method"] = "POST",
                  [":path"] = "/validate",
                  [":authority"] = "auth.local",
                  ["path"] = path,
                  ["method"] = method
                }

                local body = response_handle:body()

                if body == nil or body == '' then
                  body = "0"
                end

                jsonBody = tostring(body:getBytes(0, body:length()))

                local response_headers, response_body =
response_handle:httpCall("outbound|8787||auth.local", authentication_request, jsonBody, 100)
                if response_headers[":status"] ~= "200" then
                  response_handle:headers():replace(":status", "403")
                end
              end
              function envoy_on_request(request_handle)
                path = request_handle:headers():get(":path")
                method = request_handle:headers():get(":method")
              end
```

1. Добавление конфигурации приложения (пример)

ConfigMap: App

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-config
  labels:
    app: validator
data:
  app.yaml: |
    port: 8787
    log_level: info
    schemas:
      - name: 'foo'
        method: POST
        path: '/ping'
        schema: /schemas/foo.json
        type: 'json'
        error_response: '{"x-request-id": "{{get .Header "x-request-id"}}", "error": "{{get
.Error "error_message"}}"}'
        error_headers:
          - key: foo-bar-x
            value: foo.bar
          - key: foo
            value: '{{get .Header "foo"}}'
      - name: bar
        method: POST
        path: '/'
        schema: /schemas/bar.xsd
        type: 'xml'
        error_code: 401
        headers:
          - key: x-foo-first
            value: foo1
      - name: pass-path
        method: GET
        path: '/json'
        schema: ''
        type: 'pass'
```

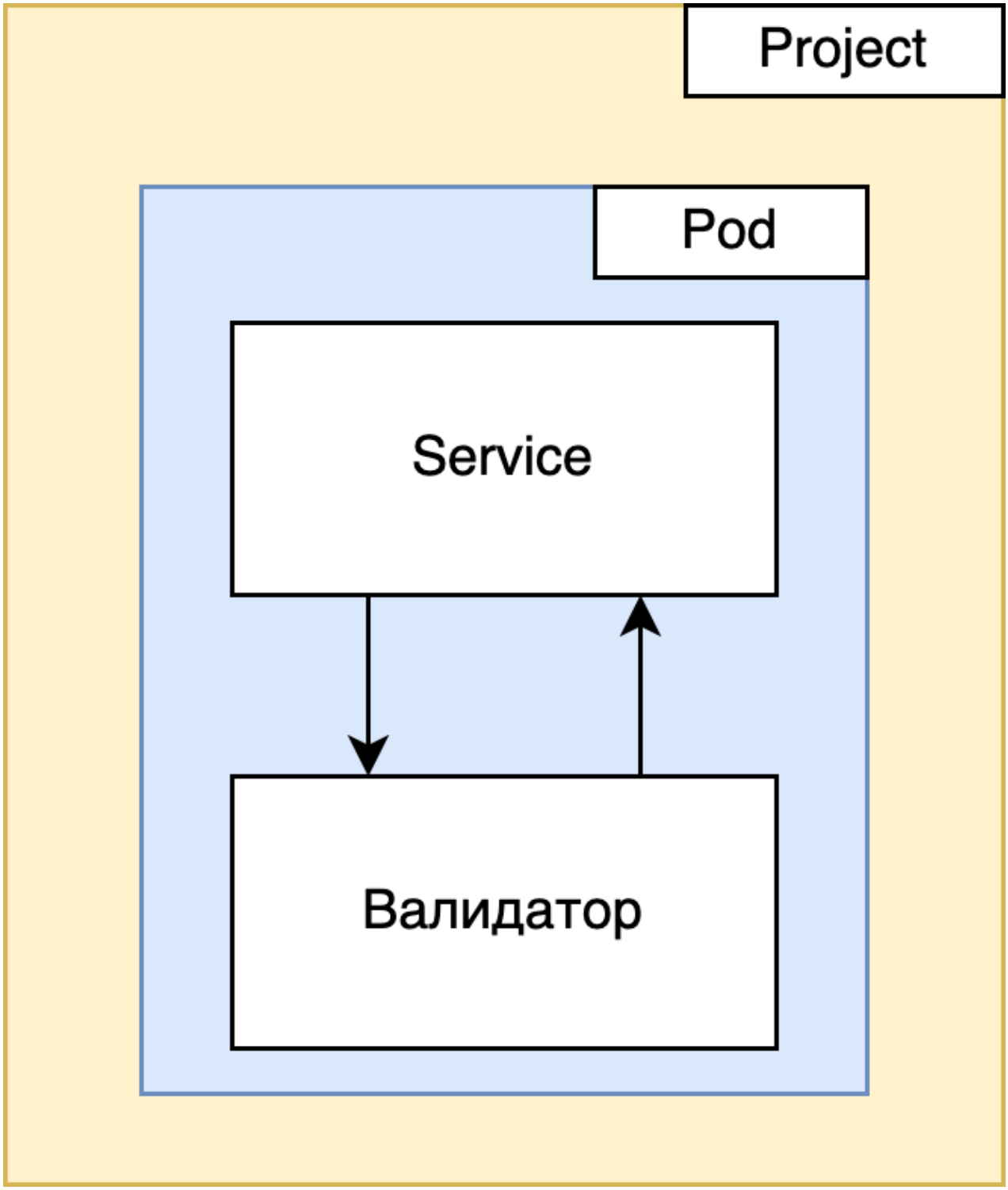
1. Добавление конфигурации схем (пример)

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-schemas
  labels:
    app: ${APP_NAME}
data:
  foo.json: |
    {
      "$id": "http://json-schema.org/draft/2019-09/json-schema-core.html",
      "$schema": "https://json-schema.org/draft/2019-09/schema",
      "$comment": "sample comment",
      "title": "Config dump",
      "type": "object",
      "properties": {
        "configs": {
          "type": "array",
          "items": {
            "type": "object",
            ...
    }
  bar.xsd: |
    <?xml version="1.0" encoding="UTF-8" ?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="address">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="street" type="xs:string"/>
                <xs:element name="street-number" type="xs:string"/>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="zip" type="xs:string"/>
                <xs:element name="country" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Схема валидации входящих запросов Валидатора

Валидация запросов происходит на Валидаторе, установленным как сайдкар к сервису. Валидация входящих запросов производится согласно предустановленным схемам валидации.



- Добавление контейнера с сервисом валидации Сервис валидации устанавливается в Deployment в раздел containers.

{REGISTRY_LINK} - подставить актуальную версию docker образа сервиса валидации.

Ingress Deployment: Containers

```
containers:
- name: validator
  resources:
```

```
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 300m
      memory: 300Mi
  terminationMessagePath: /dev/termination-log
  env:
    - name: PROJECT_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
  ports:
    - containerPort: 50051
      protocol: TCP
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - name: validator-schemas
      readOnly: true
      mountPath: /schemas
    - name: validator-config
      mountPath: /config
  terminationMessagePolicy: File
  image: >-
    {REGISTRY_LINK}
```

- Добавление конфигураций сервиса валидатор из ConfigMap

Ingress Deployment: Volumes

```
volumes:
  - name: validator-config
    configMap:
      name: validator-config
      defaultMode: 420
  - name: validator-schemas
    configMap:
      name: validator-schemas
      defaultMode: 420
```

- Добавление конфигурации приложения (пример)

ConfigMap: App

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: validator-config
  labels:
    app: validator
data:
  app.yaml: |
    port: 8787
    log_level: info
    schemas:
      - name: 'foo'
        method: POST
        path: '/ping'
        schema: /schemas/foo.json
        type: 'json'
        error_response: '{"x-request-id": "{{get .Header "x-request-id"}}", "error": "{{get
.Error "error_message"}}"}'
        error_headers:
```

```
          - key: foo-bar-x
            value: foo.bar
          - key: foo
            value: '{{get .Header "foo"}}'
      - name: bar
        method: POST
        path: '/'
        schema: /schemas/bar.xsd
        type: 'xml'
        error_code: 401
        headers:
          - key: x-foo-first
            value: foo1
      - name: pass-path
        method: GET
        path: '/json'
        schema: ''
        type: 'pass'
```

Добавление конфигурации схем (пример)

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: json-validator
  labels:
    app: ${APP_NAME}
data:
  foo.json: |
    {
      "$id": "http://json-schema.org/draft/2019-09/json-schema-core.html",
      "$schema": "https://json-schema.org/draft/2019-09/schema",
      "$comment": "sample comment",
      "title": "Config dump",
      "type": "object",
      "properties": {
        "configs": {
          "type": "array",
          "items": {
            "type": "object",
            ...
    }
  bar.xsd: |
    <?xml version="1.0" encoding="UTF-8" ?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="address">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="street" type="xs:string"/>
                    <xs:element name="street-number" type="xs:string"/>
                    <xs:element name="city" type="xs:string"/>
                    <xs:element name="zip" type="xs:string"/>
                    <xs:element name="country" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
```

Обновление

Для обновления версии сервиса валидации необходимо обновить Deployment, сайдкаром которого является REQV. Для этого поменять версию docker-образа сервиса в разделе containers или установить релиз с обновленной версией.

Артефакты ConfigMap и EnvoyFilter являются конфигурационными для сервиса, в котором инжектирован REQV, и в случае обновления ставятся из релиза с обновленной версией компоненты REQV.

Для обновления выполните следующие шаги:

1. Создайте директорию установки.

Шаг	Действия
Создайте директорию для установки	Создайте папку. *Пример: reqv*
Разархивируйте файлы	Распакуйте в созданную папку архив с конфигурационными артефактами новой версии трейсера

1. Подключитесь к проекту.

Для подключения через консоль Платформы:

Шаг	Действия
Войдите в консольного клиента платформы	В окне командной строки в приглашении введите команды: kubectl config set-credentials /<host-alias-без-точек>: --token= kubectl config set-cluster <host-alias-без-точек>: --insecure-skip-tls-verify=true --server=https://: kubectl config set-context /<host-alias-без-точек>:/ --user=/<host-alias-без-точек>: --namespace=maximov-test --cluster=<host-alias-без-точек>: kubectl config use-context /<host-alias-без-точек>:/

1. Остановите старую версию.

Для остановки через консоль Платформы:

Шаг	Действие
Остановка	В консоли выполните команду: `kubectl scale --replicas=0 deployment/<имя Деплоймента>`

1. Сохраните артефакты действующей версии.

Для сохранения через консоль Kubernetes:

Шаг	Действия	Описание
Сохраните Деплоймент	В консоли выполните команду: `kubectl get -o yaml deployment/<имя Деплоймента> > <путь к файлу>.yaml`	
Сохраните конфигурацию	В консоли выполните команду: `kubectl get -o yaml configmaps/<имя config map> > <путь к файлу>.yaml`	
Сохраните envoyFilter	В консоли выполните команду: `kubectl get -o yaml envoyFilters/<имя envoyFilter> > <путь к файлу>.yaml`	

1. Загрузите новую версию.

Для загрузки через консоль Платформы:

Шаг	Действия	Описание
Загрузите артефакт Deployment	В консоли выполните команду: `kubectl apply -f Deployment.yml`	Артефакт, содержащий Deployment.yml.
Загрузите артефакт Configmap	В консоли выполните команду: `kubectl apply -f Configmap.yml`	Артефакт, содержащий Configmap.yaml.
Загрузите артефакт EnvoyFilter	В консоли выполните команду: `kubectl apply -f EnvoyFilter.yml`	Артефакт, содержащий EnvoyFilter.yml.

Проверка работоспособности

Запрос выполняется методом POST с обязательными заголовками: **path** и **method** , по которым будет выполнена выборка необходимой схемы валидации. Также, для корректного логирования и аудита необходимо включать заголовок **x-request-id.**

Пример запроса в сервис валидации из сервиса:

```
curl -X POST 'localhost:8787/check' \
--header 'path: /alone' \
--header 'method: POST' \
--header 'Content-Type: application/xml' \
--header 'x-request-id: xxx123xxx' \
--data-raw '<?xml version="1.0" encoding="UTF-8"?>
<address>
    <street1>Orchardroad</street1>
    <street-number>15</street-number>
    <city>Uusikaupunki</city>
    <zip>1313</zip>
    <country>Farawayistan</country>
</address>' -i
```

В ответ должен прийти код 200. Данный способ подходит для всех режимов работы REQV.

Чеклист:

Проверка корректности работы:
а) На вкладке **Pods** выберите один из подов и кликните правой кнопкой мыши.
б) На открывшейся странице выберите вкладку **Logs**. Выберите контейнер валидатора запросов и убедитесь что отсутствуют ошибки в консольном выводе.

Откат

Для отката до предыдущей версии сервиса валидации необходимо обновить Deployment, сайдкаром которого сервис является. Для этого требуется поменять версию docker-образа сервиса в разделе containers на предыдущую версию или установить предыдущую версию релиза.

Часто встречающиеся проблемы и пути их устранения

Проблема	Пути решения
Ошибки добавления схем валидации в прикладной сервис	Необходимо проверить способ добавления схем и их путей в контейнер сервиса Валидации в Deployment конфигурации
Ошибки в схеме валидации при приеме запроса	Схемы валидации в ConfigMap должны быть в формате XSD
Не найден валидатор	Запрос приходит без необходимых заголовков, указанных в документации, либо в конфиге сервиса не указаны

Проблема	Пути решения
	корректные параметры выбора схем
Отсутствие Ingress Gateway/Egress Gateway в проекте	Если в проекте отсутствуют экземпляры компонента IGEG, то необходимо проверить наличие контрольной панели Istio SE и создать Deployment с Ingress Gateway/Egress Gateway с подключением к контрольной панели
В режимах Ingress Gateway и Egress Gateway запрос не перенаправляется в сервис валидации	Проверить EnvoyFilter на предмет корректного порта, на который поступает запрос и лейблов экземпляра Ingress Gateway/Egress Gateway.

Чек-лист валидации установки

1. Проверка наличия сайдкара с сервисом валидации. Открыть Deployment Ingress Gateway/Egress Gateway в режиме просмотра и найти работающий контейнер приложения валидации.
2. Проверка конфигурации сервиса валидации. Открыть раздел ConfigMaps платформы Kubernetes и найти конфигурации с именами "validator-config" и "validator-schemas".
3. Проверить наличие артефактов с типом EnvoyFilter с именем "validator-ingress" и/или "validator-egress".