APPSENTINELS DEPLOYMENT

AppSentinels Sensor and Controller for your Application

AppSentinels.ai
contact@appsentinels.ai

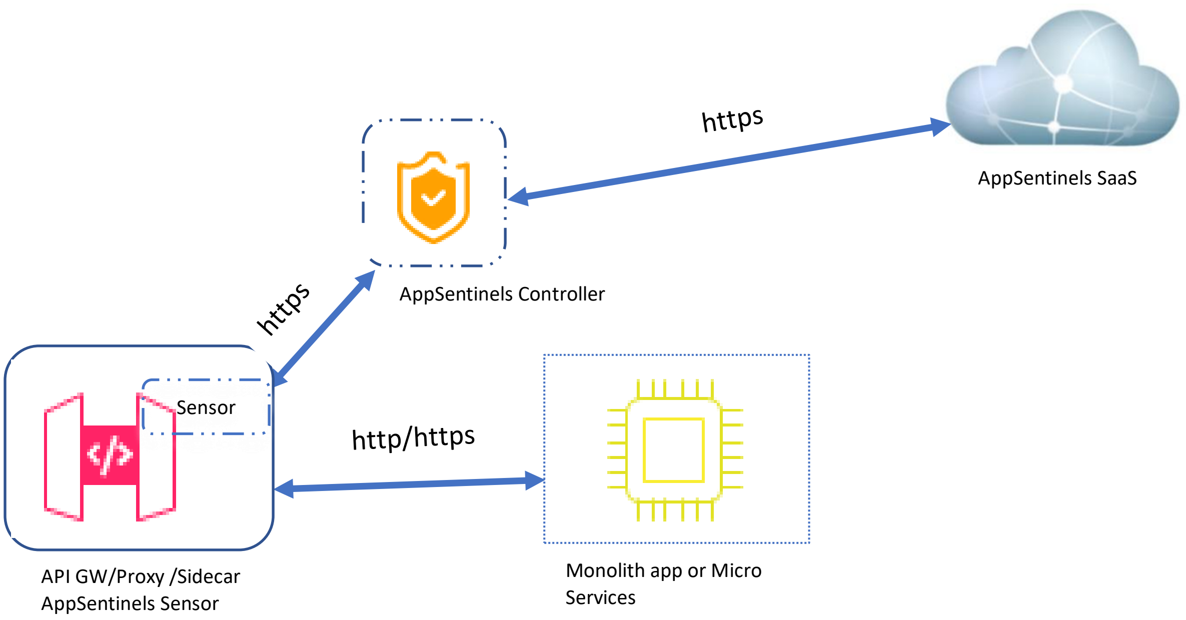
Contents

Introduction

Welcome to AppSentinels installation of Customer Premises Components.

AppSentinels supports multiple installation methods for supporting your application deployment. Any installation method takes less than 10 minutes to complete.

AppSentinels has two components that are required for installation on customer environment – a) “sensor” which is a plugin to an api-gw/proxy, and b) AppSentinels “Controller” which provides the first line of defence and communicates with AppSentinels SaaS.



A sample deployment view

AppSentinels deployment requires installing two components:

a. AppSentinels Controller
b. Sensor Installation

Sensor installation method depends on your application deployment architecture. AppSentinels supports variety of methods for deploying sensors.

AppSentinels Controller Installation

The controller performs following main functions:

- On the south side receives traffic from modules/plugin and performs local analysis and enforcement, returns local results to the sensor module
- On the north side communicates with AppSentinels SaaS Cloud, forwards the anonymized meta data, receives configuration and policies

The controller is available both docker form factor, daemon for ubuntu 18.4 or for deployment on Kubernetes environment

As Docker container on ubuntu 18.4

Step 1 – Installation of docker and prerequisites (if not available)

If docker and docker-compose (version 1.28.6 or greater) are not installed, follow the link Docker Installation

Step 2 –AppSentinels deployment directory

NOTE: Keep your API key handy. This will be either retrieved from your account on AppSentinels self-serving portal or AppSentinels support team who will share the same.

You can use a standard docker-compose file and make following changes

Image tag and hostname

```
      image: appsentinels/controller:latest
      imagePullPolicy: IfNotPresent
      hostname: appsentinels-opc-1 #or name of your choice)
```

Expose the following ports (in case of nginx)

```
      ports:
            - "9006:9006"
            - "9007:9007"
```

#for kong use "9004:9004" and "9005:9005"

Environment Variables

```
      environment:
            - SAAS_API_KEY_VALUE=<Key-Value>
            - APPLICATION_INFO=<application-name>
            - ENVIRONMENT=<production/staging> #default is production
            - SAAS_SERVER_NAME=cloud.appsentinels.ai #or one shared
by AppSentinels support team
```

(for a sample docker-compose and environment file see the section – Controller Sample Docker Compose File)

Step 3 – run the controller

```
      $ /usr/bin/docker-compose up -d
```

Step 4 – verification

- Check on dashboard "system health" tab to see the status of the "Controller"

As Daemon on ubuntu 18.4

Contact AppSentinels support team (customer-support@appsentinels.ai) and support team will help you with this deployment

On Kubernetes environment

In Kubernetes environment the controller can be deployed as a K8 POD.

Sample deployment file

```
kind: Deployment
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.21.0 (992df58d8)
  creationTimestamp: null
  labels:
    io.kompose.service: onprem-controller
  name: onprem-controller
  namespace: appsentinels
spec:
  replicas: 1
  selector:
    matchLabels:
      io.kompose.service: onprem-controller
  strategy:
    type: Recreate
  template:
    metadata:
      annotations:
        kompose.cmd: kompose convert
        kompose.version: 1.21.0 (992df58d8)
      creationTimestamp: null
      labels:
        sidecar: disabled
        io.kompose.network/onprem-mesh: "true"
        io.kompose.service: onprem-controller
    spec:
      containers:
      - env:
        - name: SAAS_SERVER_NAME
          value: dev-cloud.appsentinels.ai
        - name: SAAS_API_KEY_VALUE
          value: XXXXXXXXXXXXXXXXXXXXXXX
        - name: APPLICATION_INFO
          value: actor
        - name: NAMESPACE_INFO
          value: info
        - name: ENVIRONMENT
          value: production
        image: appsentinels/controller:latest
        imagePullPolicy: IfNotPresent
        name: onprem-controller
        ports:
        - containerPort: 9006
        - containerPort: 9007
        resources: {}
      hostname: opc
      restartPolicy: Always
      serviceAccountName: ""
```

Check on dashboard “system health” tab to see the status of the “Controller”

Contact AppSentinels support team (customer-support@appsentinels.ai) and support team will help you with this deployment

Sensor Installation

The AppSentinels sensor stays closer to the API packet path. It is quite small footprint component which is primarily responsible for:

- Passing data to the controller and taking return actions from controller
- Provide guaranteed latencies for above communication (with controller) and depending on configuration perform fail-open or fail-close

Currently, AppSentinels supports following types of sensors.

a) API-GW/Proxy plugins: If the deployment has API-GW/proxy like nginx, kong, nginx-ingress controller where the modules can be installed, this is the preferred way. This is supported for both monolithic apps as well as microservices environment. This involves a few steps depending on your server (web server or api gateway).
   The following are the supported options for API GW and nginx proxy:
   a) NGINX Plugin installation
   b) Kong Plugin Installation
   c) NGINX Ingress Controller installation (nginx ingress controller for Kubernetes environment)

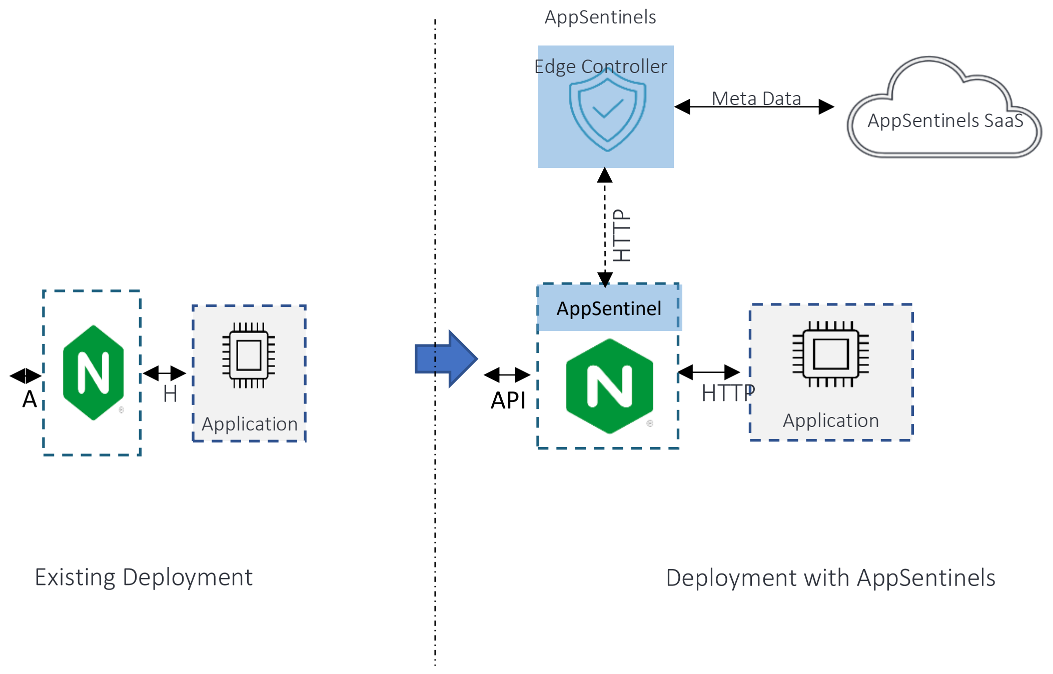
b) Inline HTTP Proxy: Module as HTTP proxy in between the server/gateway and the application. In this case a single container/vm (which running a reverse-proxy and Controller) is logically wired between gateway and the application. See HTTP Proxy Mode

c) Network Tap mode: See Tap Mode Plugin Installation

d) Serverless AWS Lambda installation: See AWS Lamda Plugin Installation

e) Side Cars for East west traffic analysis: This is an add-on to monitor east-west traffic in Kubernetes environment. Here custom-envoy (module) as a sidecar in pods. See Installation on Kubernetes for East-West Traffic Analysis

NGINX Plugin installation



AppSentinels provides NGINX Plugins for specific distributions and versions of NGINX.

Step 1 – Check and download the nginx version

- To retrieve your version details of your currently nginx, run the following command
  ```
  $ nginx -V
  ```
- Check with the list of supported plugins
  - The list of plugins available along with download links is available at https://storage.googleapis.com/appsentinels-deployment/nginx/supported-versions.txt
  - If you do not find your version and distribution in the supported list OR you are using custom nginx contact customer-support@appsentinels.ai

- Download and install
  - download the version corresponding to your running version (latest link to be picked from https://storage.googleapis.com/appsentinels-deployment/nginx/supported-versions.txt
  - $ curl -v https://storage.googleapis.com/appsentinels-deployment/nginx/ubuntu-18.04-nginx-1.18.0.tar.gz --output ubuntu-18.04-nginx-1.18.0.tar.gz
  - untar the file $ tar -zxvf nginx-module-<os-dis>-<nginx-version>.tar.gz example $ tar -zxvf nginx-module-18.04-1.18.0.tar.gz
  - $ sudo cp 1.18.0/*.so /etc/nginx/modules/

Step 2 – Update the config

- NOTE: Keep the name and/or IP address of the vm/machine where the controller is running handy. In below config it is referred as opc_server (the name should be resolvable on the network)update nginx.conf
  - the config file by default is at `/etc/nginx/nginx.conf` (required sudo permission)

- add these 3 lines in the global section above the pid file information

```
load_module /etc/nginx/modules/nginx_ext_auth_module.so;
load_module /etc/nginx/modules/nginx_ext_access_log_module.so;
thread_pool ext_access_log_thread_pool threads=1 max_queue=65535;
```

- Typical configuration,
  Inside the **server and location** blocks copy the following config

```
#Your server block
server {
listen .....;

     #Appsentinels block [
        ext_access_log_server http://opc_server:9007;
        ext_access_log_api /nginxlog;
        ext_auth_fail_allow on;

        location /auth {
            internal;
            proxy_set_header X-Req-Uri $request_uri;
            proxy_set_header X-Method $request_method;
            proxy_set_header X-Req-Host $host;
            proxy_set_header X-Client-Addr $remote_addr:$remote_port;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            proxy_connect_timeout 500ms;
            proxy_read_timeout 500ms;
             proxy_set_header Connection "";
             proxy_http_version 1.1;
            proxy_pass http://opc_server:9006;
        }

 location <your-url-1> {
ext_auth_request /auth;
.....
 }
 location <your-url-2> {
ext_auth_request /auth;
.....
 }

        #appsentinels block ]
}
```

- In case of URL specific configurations,
  Inside the **server and location** blocks copy the following config

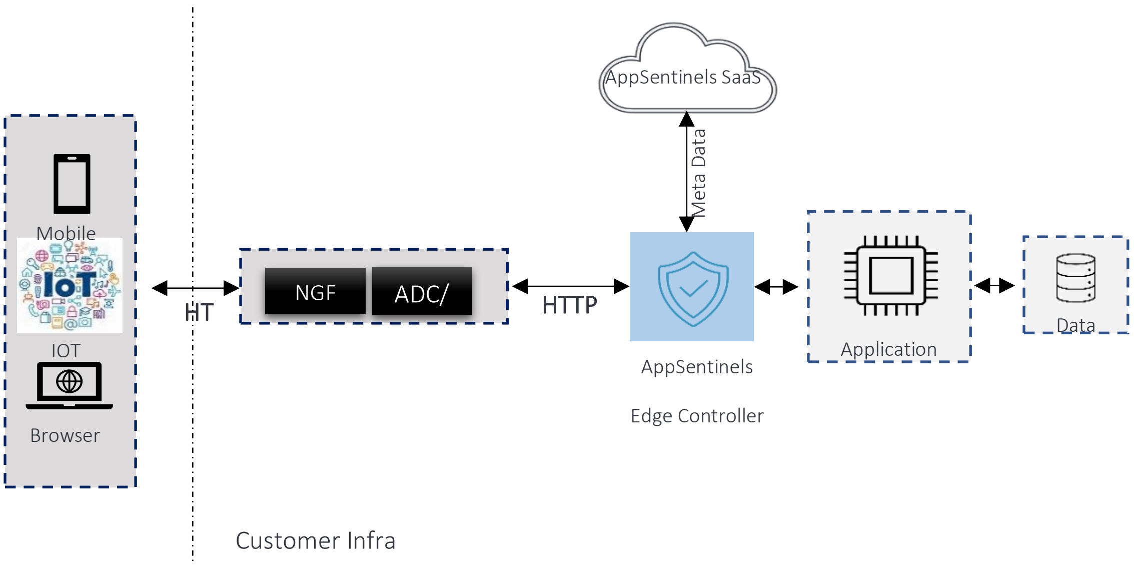
```
#Your server block
server {
listen .....;

       #Appsentinels block [
        ext_auth_fail_allow on;

        location /auth {
            internal;
            proxy_set_header X-Req-Uri $request_uri;
            proxy_set_header X-Method $request_method;
            proxy_set_header X-Req-Host $host;
            proxy_set_header X-Client-Addr $remote_addr:$remote_port;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            proxy_connect_timeout 500ms;
            proxy_read_timeout 500ms;
             proxy_set_header Connection "";
             proxy_http_version 1.1;
            proxy_pass http://opc_server:9006;
        }

location <your-url-1> {
               ext_auth_request /auth;
      ext_access_log_server http://opc_server:9007;
               ext_access_log_api /nginxlog;
       .....
}
       location <your-url-2> {
               # No inspection needed for this url
       ....
}

        #appsentinels block ]
}
```

- Enforce the config
  `$ sudo service nginx restart`

Step 3 – Verification

- Run some traffic
- Check the dashboard for your API's in the API Catalogue

HTTP Proxy Mode or Inline Mode



In this mode a proxy is placed in front of the packet path. The proxy receives the traffic sends the logs to the AppSentinels controller and forward the traffic to the application. AppSentines use envoy based HTTP proxy, which is a lightweight proxy recommended as part of this deployment solution.

Step 1 – Prerequisite

If docker and docker-compose (version 1.28.6 or greater) is not installed, follow the link Docker Installation

Step 2 – Pull AppSentinels deployment directory

NOTE: Keep your API key handy. This will be either retrieved from your account on AppSentinels self-serving portal or AppSentinels support team who will share the same.

- $ mkdir appsentinels; cd appsentinels
- $ curl -v https://storage.googleapis.com/appsentinels-deployment/http-proxy/monolith.tar.gz
- $ tar -zxvf monolith.tar.gz
- the directory structure looks like

```
appsentinels/monolith
├── docker-compose-production.yaml
├── env
│   ├── env-envoy.txt
│   └── env.txt
├── readme.txt
└── utils
    ├── dp-commands.sh
    ├── set-core-config-settings.sh
    └── set-syslog.sh
```

- $ cd appsentinels/monolith
- update the following parameters in env/env.txt (only SAAS_API_KEY_VALUE is mandatory, rest are used for visibility)

```
SAAS_API_KEY_VALUE=<your-api-key>
APPLICATION_INFO=<>your application name>
NAMESPACE_INFO=<namespace info>
ENVIRONMENT=<production/staging>
```

- update the following parameters in env/env-envoy.txt

```
#upstream service name/application
ENVOY_SERVICE_NAME=<application-service-name>
#this is the port on which application is running
ENVOY_SERVICE_PORT=<application-service-port>
#this is the external port on which traffic will be coming
ENVOY_EXTERNAL_DATA_LISTENING_PORT="8000"
```

- pull the images

```
$ /usr/bin/docker-compose --env-file env/env-envoy.txt pull
```

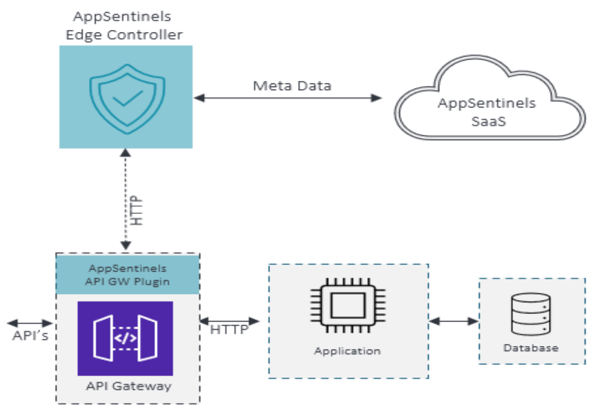
Step 3 – Run the controller

- `$ sudo sh set-syslog.sh`
- `$ /usr/bin/docker-compose --env-file env/env-envoy.txt up -d`

Step 4 – Verification

- Check on dashboard “system health” tab to see the status of the “Controller”

Kong APIGW Plugin Installation



Step 1: Download & Build the Kong Plugin

- `$mkdir appsentinels; cd appsentinels`
- `$ curl -v  https://storage.googleapis.com/appsentinels-deployment/kong/kong-plugin.tar.gz` (Please reach out to customer-support@appsentinels.ai)
- `$ tar -zxvf kong-plugin.tar.gz`
    - the directory structure will be like

```
appsentinels/kong-plugin-http-log-with-body
├── kong-plugin-http-log-with-body-0.1.1-2.all.rock
├── sample-Docker-compose
```

The Kong in your environment can be running in either "VM" or a "docker" form factor. Follow the below steps based on deployment

Kong on VM

- Installing the plugin (Ref:- https://docs.konghq.com/gateway-oss/2.4.x/plugin-development/distribution/)
    - `$ luarocks install kong-plugin-http-log-with-body-0.1.1-2.all.rock -local`
        - The plugin will be installed at `$HOME/.luarocks/share/lua/5.1/kong/plugins/http-log-with-body`
- Update the plugin section in kong configuration to add this plugin
  `plugins = bundled,http-log-with-body`
- Execute the following commands – $ kong prepare; kong load
- On the admin interface of Kong (default 9501) enable the plugin
    - `$ curl -X POST http://localhost:9501/plugins/ --data name=http-log-with-body --data config.http_endpoint=http://<controller-name>:9004/konglog`
      where controller-name is the name of the host where the AppSentinels controller is running

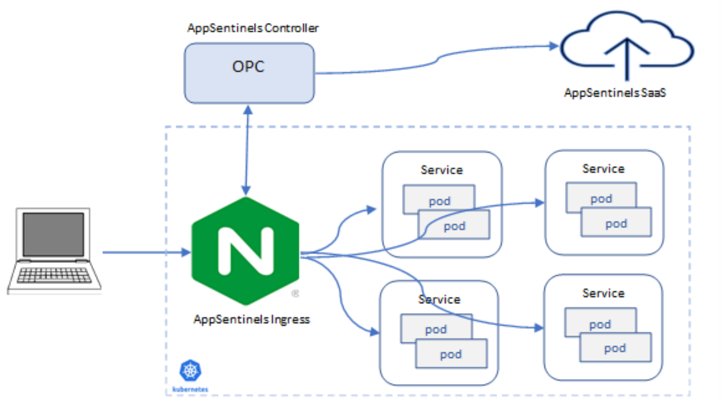
Kong On Docker

- Refer to sample-Docker-compose for below steps
- Package rock file into kong's Dockerfile
  ADD kong-plugin-http-log-with-body/kong-plugin-http-log-with-body-0.1.1-2.all.rock /tmp/
- Install it locally as root user doesn't exist in kong docker image (https://github.com/Kong/docker-kong/issues/328)
  - RUN `luarocks install /tmp/kong-plugin-http-log-with-body-0.1.1-2.all.rock --local`
- Build the kong image
  - Enable the plugin via docker-compose.yaml by adding it to $KONG_PLUGINS environment variable
    - `KONG_PLUGINS: bundled,http-log-with-body`
- Deploy the docker image
- On the admin interface of Kong (default 9501) enable the plugin

  -
    ```
    $ curl -X POST http://localhost:9501/plugins/ --data
    name=http-log-with-body --data
    config.http_endpoint=http://<controller-
    name>:9004/konglog
    ```
    where controller-name is the name of the host where the AppSentinels controller is running

NGINX Ingress Controller installation for Kubernetes

This requires deployment of AppSentinels NGINX ingress controller in K8.



Step 1: Image updates

Update the image link to "*appsentinels/nginx-ingress-controller:latest*" in your deployment file. Once updated redeploy the ingress

```
…
containers:
      - name: controller
               - image: appsentinels/nginx-ingress-
controller:latest
      - imagePullPolicy: IfNotPresent
…
```

Step 2: Configuration updates

```
      apiVersion: networking.k8s.io/v1beta1
      kind: Ingress
```

```
      metadata:
        name: juice-shop-ingress
        annotations:
           kubernetes.io/ingress.class: "nginx"
           nginx.org/location-snippets: |
           ext_auth_request /opc-auth;
           nginx.org/server-snippets: |
           ext_auth_fail_allow on;
           location /opc-auth {
                internal;
                proxy_set_header X-Req-Uri $request_uri;
                proxy_set_header X-Method $request_method;
                        proxy_set_header X-Req-Host $host;
                        proxy_set_header X-Client-Addr
$remote_addr:$remote_port;
                        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
                        proxy_set_header X-Forwarded-Proto $scheme;
                        proxy_connect_timeout 10ms;
                        proxy_read_timeout    500ms;
                        # OPC hostname or IP where auth listening is
present
                        proxy_pass http://<appsentinels-
controller>:9006;
                   }
                   ext_access_log_server http://<appsentinels-
controller>:9007;
                   ext_access_log_api /nginxlog;
```

Step 3: Apply the configuration and check the pod-status

```
$ kubectl apply -f deployment/kubernetes/nginx-ingress-
controller/ingress.yaml
```

Sidecar on Kubernetes for East-West Traffic Analysis

Injecting sidecar on each application pods requires a) sidecar injector deployment, and pulling AppSentinels sidecar image.

AppSentinels sidecar injector

Sample deployment yaml and utility scripts are available at https://storage.googleapis.com/appsentinels-deployment/k8-sidecar/k8-sidecar.tar.gz . Please contact contact@appsentinels.ai for access.

1. Create a separate namespace
    a. `$ kubectl create ns appsentinels`
    b. `$ kubectl label ns appsentinels ns-skip-inject=yes //To avoid mutating the webhook server`
2. Generate secrets TLS for the webhook server
    a. Use openssh commands to generate public/private certificates
    b. Or, use AppSentinels utility script to generate the certificates.

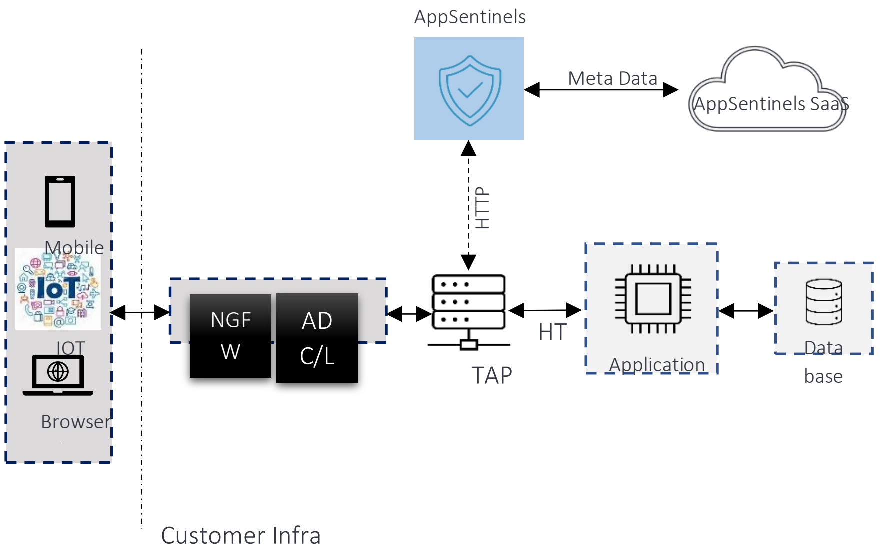
       ```
       $ generate-cert.sh && ./generate-cert.sh
       ```

3. Deploy TLS secrets
    a. `$ kubectl --namespace=appsentinels create secret tls webhook-certs --cert=keys/server.crt --key=keys/server.key`
4. Deploy the sidecar injector
    a. `$ kubectl apply -f deployment.yaml`
    b. `$ kubectl apply -f webhook.yaml`

By default the webhook will try to push the sidecar to all pods in all namespaces this can be disabled using the below two methods.

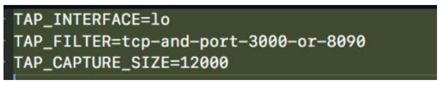
- To skip side-car injection in the pod add `sidecar: disabled` key value in the deployment yaml of the pod as labels under (metadata>labels) and (spec>template>metadata>labels)
- To skip the whole namespace you can put label (`ns-skip-inject:'yes'`) for the namespace.

Tap Mode Module Installation



To deploy TAP mode, no separate sensor installation is required. Network TAP need to be directed to AppSentinels controller. The AppSentinels controller need these additional configurations to access TAP network inputs.

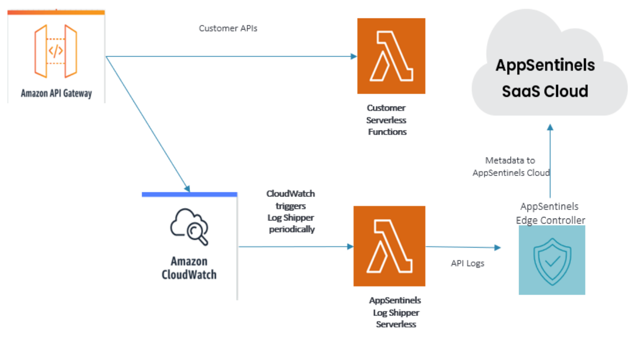
1. If running inside docker, host networking needs to be enabled.
2. Following additional environment variables need to be passed.



```
TAP_INTERFACE=lo
TAP_FILTER=tcp-and-port-3000-or-8090
TAP_CAPTURE_SIZE=12000
```

    a) Set TAP_INTERFACE to the interface where network TAP input is sent
    b) Ports of HTTP traffic set to TAP_FILTER

AWS Lambda Module Installation


Customer APIs
Amazon API Gateway
Customer Serverless Functions
AppSentinels SaaS Cloud
Metadata to AppSentinels Cloud
CloudWatch triggers Log Shipper periodically
Amazon CloudWatch
AppSentinels Log Shipper Serverless
API Logs
AppSentinels Edge Controller

This is recommended for serverless application protection. The leverages AWS APIGW CloudWatch logs. Contact AppSentinels support team (customer-support@appsentinels.ai) and support team will help you with this deployment

Appendix A

Docker Installation

Run these commands one at a time

- `$ sudo apt install curl; sudo apt install jq`
- `$ curl -fsSL https://get.docker.com -o get-docker.sh`
- `$ sudo apt install docker-compose`
- `$ sudo curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/bin/docker-compose`
- `$ sudo adduser <user-name> docker`
- `logout and login-in again (the addition of user to group docker is active only after re-login)`

Controller Docker Compose Sample

docker-compose.yaml content

```
version: "3.3"
services:
  onprem_controller:
    container_name: onprem-controller
    image: appsentinels/controller:latest
    hostname: appsentinels-opc-1
    environment:
      - SAAS_API_KEY_VALUE=<api-key>
      - APPLICATION_INFO=<application-name>
      - ENVIRONMENT=production
    ports:
      - "9006:9006"
```

```
        - "9007:9007"
        - "127.0.0.1:9101:9101"
      logging:
        driver: syslog
        options:
          tag: appsentinels-onpremcontroller
      volumes:
        - /var/log/appsentinels/:/var/crash
        - /var/crash:/var/crash
        - /var/log/appsentinels:/var/log/appsentinels
```

Run the controller

- `$ /usr/bin/docker-compose pull  && /usr/bin/docker-compose up -d`