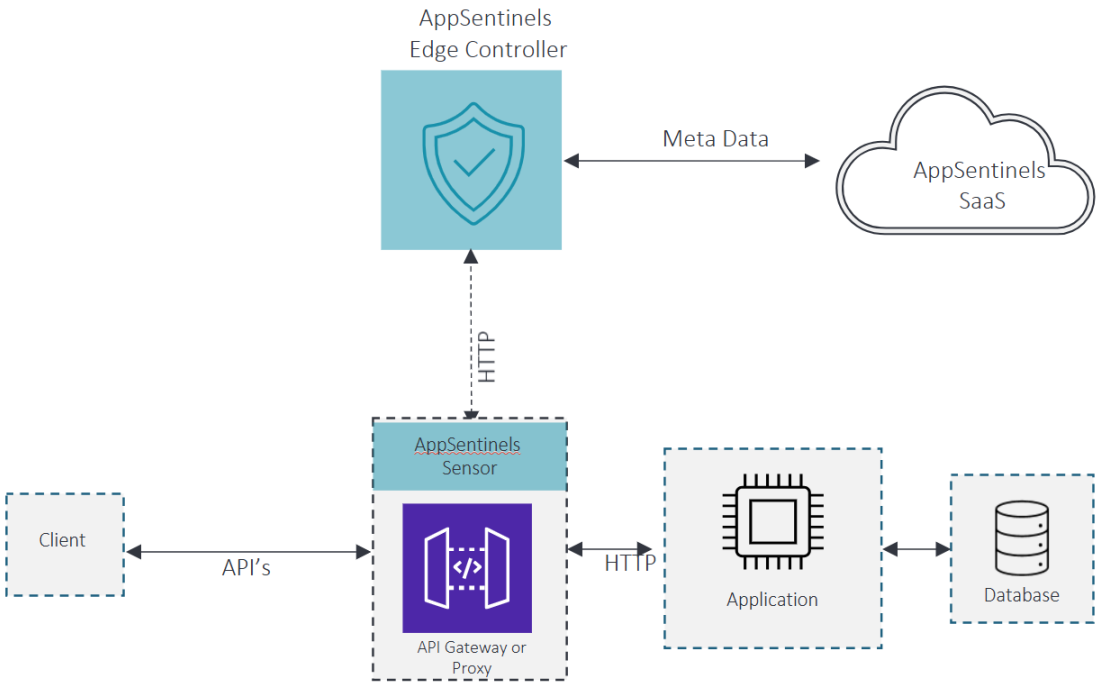
AppSentinels Edge Controller Pre-Requisites & Deployment

Contents

Revision	Date Modified	Author	Comments
1.0	08-Dec-23		Initial Draft
1.1	01-Aug-24	Sachin	Created consolidated controller deployment document
1.2	12-Jun-2025	Sachin	Horizontal scaling

1. Introduction

AppSentinels Edge Controller processes the API traffic data received from the Sensor and forwards the meta data to the Cloud Platform for AI/ML processing. The Edge Controller detects the Security Events and takes preventive action using Firewall/API GW.



2. Pre-requisites

Operating System and Hardware Requirements: -

- Operating System: Ubuntu 22.04 or REDHAT 8.6
- CPU: 4 cores x86_64*
- RAM: 16G of RAM*
- 50 GB free disk space in /var partition

* CPU & Memory requirement will change based on API traffic rate and models enabled for detection. Approximately, a single core of controller will allow for ingesting of 500-1000 API logs per second. The above number will serve between 2000 to 4000 API logs per second.

Packages Required: -

- docker version 23.0 or higher
- docker-compose version 1.28.6 or higher (if deploying with docker-compose)

Network Connectivity Requirements: -

- Outing TCP Port 443 to in-cloud.appsentinels.ai or in-cloud-s.appsentinels.ai (for POC) should be allowed to send data to the AppSentinels Cloud
- The TCP port range 9002-9007 should be opened for the AppSentinels Sniffer Sensor/Plugin to send traffic logs to the AppSentinels Controller. The specific port used will depend on the integration between the Sensor/Plugin and the AppSentinels Controller.

- Access to *.docker.io to download the Docker images. If *.docker.io can't be allowed, access has to be provided to the following domains
    - docker.io
    - auth.docker.io
    - registry-1.docker.io
    - production.cloudflare.docker.com

3. Deployment Options

1. Docker-compose

Deploy AppSentinels Edge Controller as docker container by updating the below YAML with values specific to application environment.

```
version: "3.3"
services:
  ng-edge-controller:
    container_name: ng-edge-controller
    restart: on-failure:5
    image: appsentinels/ng-controller:latest
    hostname: ng-edge-controller-1
    environment:
      - APPLICATION_DOMAIN=<your-app-domain>      # Add the name of application or application group, probably something to
                                                  # denote a bunch of apps
      - ENVIRONMENT=<environment>                 # Current environment, qa va dev vs prod vs staging
      - SAAS_SERVER_NAME=<your allocated saas server>  # Hostname of allocated saas server
      - SAAS_API_KEY_VALUE=<your provided api key>     # AppSentinels provided API key for communication to saas
    ports:
      - "9004:9004"                                       # default for merged logs, but will depend on sensor
                                                          #     9002:9002 for sensors like envoy
                                                          #     9004:9004 for merged log sensors (eg: kong, lambda,
sniffer)
                                                          #     9006:9006 for sensors like nginx or apigee
       - HTTPS_INSECURE_SKIP_VERIFY=false                  # input needed here, set to true if
                                                           # using self signed certs for onprem deployments



    logging:
      driver: local
      options:
        max-size: 10m
    volumes:
      - /var/crash/:/var/crash
```

Reference spec: https://sample-config.appsentinels.ai/appsentinels-deployment/sample-docker-compose/docker-compose-kong.yaml

Secure Logging:

Edge Controller can communicate to sensors over HTTPS. To configure this, controller will need to be provided with server certificate and private key. These can be mounted at locations /certs/server.crt and /certs/server.key respectively.

```
  ng-edge-controller:
....
    volumes:
      - ./server_public.pem:/certs/server.crt
      - ./server_private.pem:/certs/server.key
```

2. Installation script

AppSentinels Edge controller can also be deployed in a zero touch manner on VM instances. AppSentinels provides an installation script that can be inserted into during the first launch of the instance. This script installs the AppSentinels Edge Controller service on Ubuntu, Amazon Linux or RHEL/CentOS along with necessary utils like docker (if not present).

If inserting during first run is not feasible, this script can run in a standalone fashion as well.

2.1 AWS EC2 support

AWS provides for user data configuration during AMI launch. AppSentinels installation script can be inserted here. This allows for installation of required tools or packages during the first boot of an AMI.

2.2 Script Configuration:

Depending on the deployment and the application running, the following basic configurations will need to be done.

```
CONTAINER_NAME="appsentinels-edge-controller"          #input needed here
HOSTNAME=$(hostname)                                   #input needed here, needs to be unique across all the edge
controllers
APPLICATION_DOMAIN="<your application domain>"         #input needed here, probably something to denote a bunch of apps
ENVIRONMENT="prod"                                     #input needed here, what is the current environment, qa vs prod vs
dev etc
SAAS_SERVER_NAME="<your allocated saas server>"        #input needed here
SAAS_API_KEY_VALUE="<AppSentinels provided API key>"   #input needed here
CONTAINER_PORT_MAPPING="9004:9004"                     #input needed here, default for merged logs, but will depend on
sensor
                                                       #     9002:9002 for req and resp logging sensors like envoy
                                                       #     9004:9004 for merged log sensors (eg: kong, lambda, sniffer)
                                                       #     9006:9006 for req/resp logging sensors (eg: nginx, apigee)
HTTPS_INSECURE_SKIP_VERIFY="false"                     #input needed here, set to true if using self signed certs
                                                       #for SAAS
```

Script Location:
https://sample-config.appsentinels.ai/appsentinels-deployment/edge-controller/install-appsentinels-controller.sh

3. Kubernetes service

AppSentinels controller can also be run as a Kubernetes service. The service requires the environmental variables to be defined as in earlier section (i.e in docker-compose). In addition, a container port(s) specific to sensors will need to be opened up.

Below is one such example of a spec where controller is running as a deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    io.kompose.service: appsentinels-controller
  name: appsentinels-controller
  namespace: appsentinels
spec:
  replicas: 1
  selector:
    matchLabels:
      io.kompose.service: appsentinels-controller
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: appsentinels-controller
        io.kompose.service: appsentinels-controller
    spec:
      # ########################################################
      # Start of properties of appsentinels controller container
      containers:
      - env:
        - name: SAAS_SERVER_NAME
          value: <your allocated saas server hostname>
        - name: SAAS_API_KEY_VALUE
          value: <Your API key>
        - name: APPLICATION_DOMAIN
          value: <Your app domain>
        - name: ENVIRONMENT
          value: <Your environment>
        image: appsentinels/ng-controller:latest
        imagePullPolicy: IfNotPresent
        name: appsentinels-controller
        ports:
        - containerPort: <sensor specific port>
        resources:
          requests:
            cpu: 4
            memory: 8Gi
          limits:
            cpu: 4
            memory: 16Gi
      hostname: appsentinels-controller # This is mandatory
      # end of properties of appsentinels controller container
      # ########################################################
      restartPolicy: Always
```



```
      serviceAccountName: ""
      dnsPolicy: "None"
      dnsConfig:
         nameservers:
           - 8.8.8.8 #UPDATE REQUIRED
status: {}
```

Reference spec:

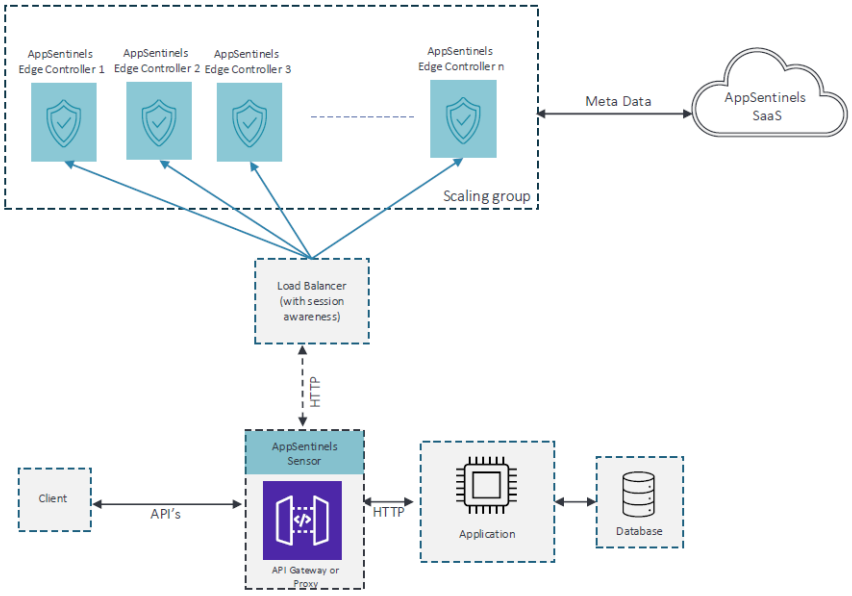
https://sample-config.appsentinels.ai/appsentinels-deployment/k8/controller/edge-controller.yaml

There are other modes of deployment in Kubernetes and can be realized based on the needs. Please get in touch with AppSentinels support team to take it forward.

4. Horizontally Scaled Deployment

To ensure high availability, scalability, and resilience, AppSentinels Edge Controllers can be deployed in a horizontally scaled architecture.

Architecture Overview



In a horizontally scaled deployment, multiple AppSentinels Edge Controllers are deployed in a stateless fashion behind a session-aware load balancer or ingress controller. This architecture allows the system to handle increased throughput and user sessions reliably.

Key Characteristics

- Controllers are identical in terms of resources and policy configuration.

- Stateless design necessitates session stickiness to maintain user context.

- Session stickiness is enforced via user session information available in headers (such as XFF, authorization header etc)

- Horizontal scaling is achieved via Kubernetes Deployments.

- Deployment is compatible with both Out-of-Band (OOB) and enforcement modes.

Request Flow

1. Clients send traffic to a CDN, WAF, proxy or gateway configured with AppSentinels Sensors.

2. Sensors capture HTTP API traffic and forward logs to an Ingress Controller over HTTPS. Sensors also capture session information in **X-As-Queue-Hash** header.

3. The Ingress Controller uses session-specific information (**X-As-Queue-Hash**) to enforce sticky routing, ensuring logs from the same user or session are routed to the same Edge Controller instance.

4. Each Edge Controller processes the logs and performs analysis and enforcement as needed.

Edge Controller Load Balancer Requirements

- Must support session stickiness based on headers. A session-aware ingress controller/load balancer (e.g., NGINX Ingress with nginx.ingress.kubernetes.io/upstream-hash-by) is mandatory.

- Can be a standard load balancer or a Kubernetes ingress controller with session-affinity annotations.

- Should health-check each controller instance to prevent routing to unhealthy nodes.

Sample Configuration

To configure NGINX Ingress Controller to perform session stickiness (hash-based load balancing) using the X-As-Queue-Hash header, you use the annotation:

```
nginx.ingress.kubernetes.io/upstream-hash-by: "$http_x_as_queue_hash"
```

The above annotation tells NGINX to hash the value of the X-As-Queue-Hash header to determine backend selection.

This ensures all requests with the same **X-As-Queue-Hash** value go to the same controller pod, preserving context and log correlation even in a stateless horizontally scaled environment.

5. Verify Deployment

AppSentinels Edge Controllers deployed in your environment will be listed on the System Health page on AppSentinels Dashboard.

Generate application traffic in the monitored application and check the API catalogue on AppSentinels Dashboard for the discovered APIs.