Apigee Edge: Introduction to Microgateway

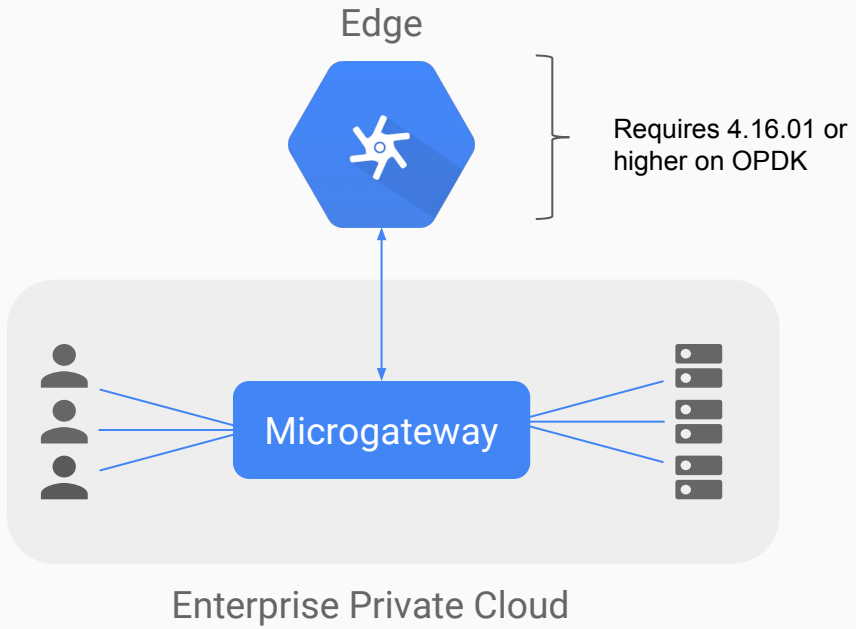
Srinandan Sridhar (srinandans@)

Objective

> Deploy a lean API runtime infrastructure in your private enterprise environment while getting all the benefits of Edge API management in the cloud

Use cases

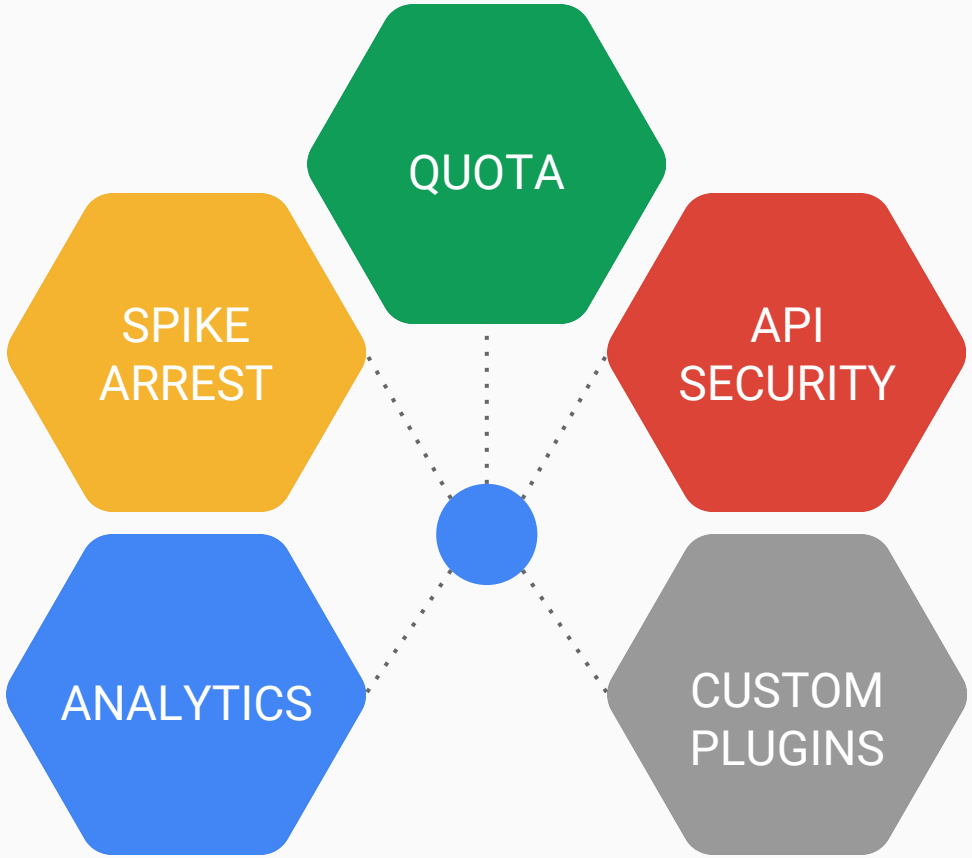
- Reduce latency of traffic for close proximity services
- Keep API traffic within the enterprise-approved boundaries because of security or compliance purposes
- Continue processing messages if internet connection is lost



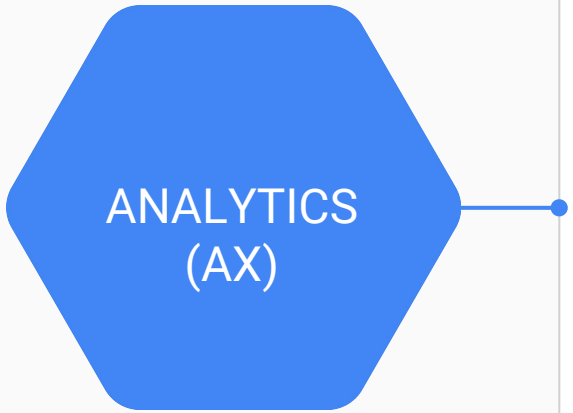
Description

- Microgateway is a Node.js application and can be run by anyone with permission to run such applications on a given machine
- It’s a lightweight process that can run close to the target API—even on the same machine
- It requires Node.js 4.5 LTS or later and runs on Windows, Linux, and macOS

Features

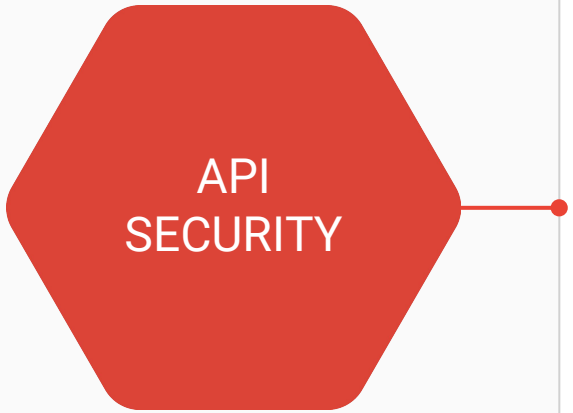


Features



- AX data gets sent to Edge asynchronously in batch
- If the AX data cannot be sent and the maximum batch size is reached, analytics data is dropped
- AX data is sent, regardless of the target availability

Features

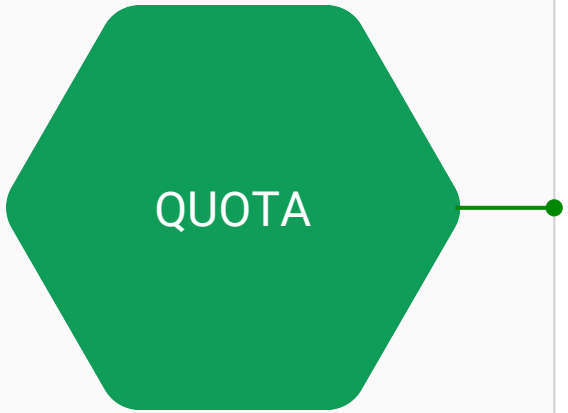


- API Key
- IP Filtering (whitelist, blacklist, or both)
- OAuth
  - JWT-based tokens
  - Default OAuth is client credentials grant type
  - Password grant can be used if IdP is configured

Features

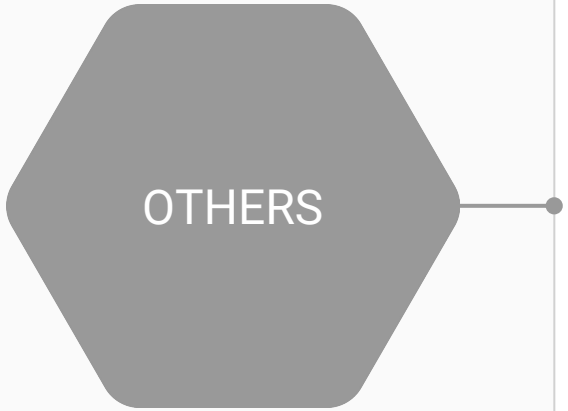
- Configured per Microgateway instance
- Enforced 100% and in memory

Features



- Distributed quota (across MG instances)
- Configured on Edge at product level
- Asynchronous request to Edge to update count
- Quotas are approximate

Features



- Support for one-way and two-way TLS from consumer to microgateway and microgateway to backend
- Transformation between JSON to XML (and vice versa)
- Support for forward-proxy (to Apigee Cloud and backend API endpoints)
- Default streaming-enabled

Edge Enterprise Gateway vs. Microgateway

Feature	Enterprise Gateway	Microgateway
OAuth: JWT tokens • Supports password and client credentials grant	✓	✓
OAuth: opaque tokens • Supports all four grant types	✓	✗ *
Spike Arrest & Quota	✓	✓
Caching	✓	✗ *
Threat protection (XML, JSON, RegEx)	✓	✗ *
Callout (Python, Java, XSLT)	✓	✗
JavaScript callouts, JSON-XML transformation	✓	✓
Bot Detection & Monetization	✓	✗
Containerization or Cloud Foundry Deployments	✗	✓
Logging & Monitoring	✓	file system only; basic monitoring
Analytics	✓	✓ customer dimensions are not possible

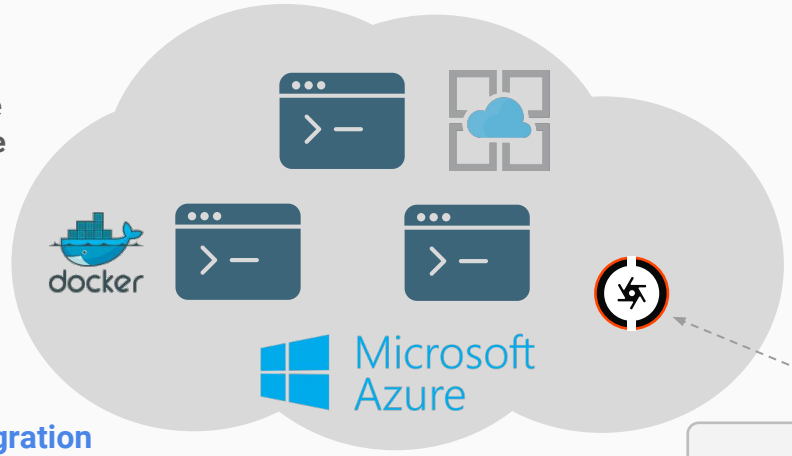
* Possible via customization

When to use Microgateway?

Requirement	Which gateway?
• Building new APIs or new microservices • Deploying in PaaS (Cloud Foundry or containers/Kubernetes)	Microgateway
• Distributed API management (a gateway for each API or a few APIs)	Microgateway
• A single (or a few) gateways for the enterprise • Gateway pattern	Enterprise Gateway
• Complex mediation rules (API orchestration, aggregation, etc.)	Enterprise Gateway
• Legacy services (such as SOAP) modernization	Enterprise Gateway
• Internal APIs or APIs for application-to-application integration (simple APIM policies)	Microgateway
• External API traffic • Monetization	Enterprise Gateway
• Scale: 100s of APIs per second	Microgateway
• Scale: 1000s of APIs per second	Enterprise Gateway

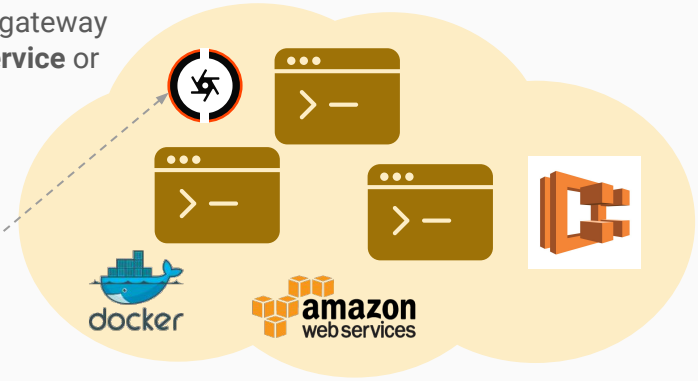
Deep integrations with multiple clouds

Apigee on Azure Cloud
Easily deploy Apigee Edge Microgateway as an **Azure App Service** or **docker**

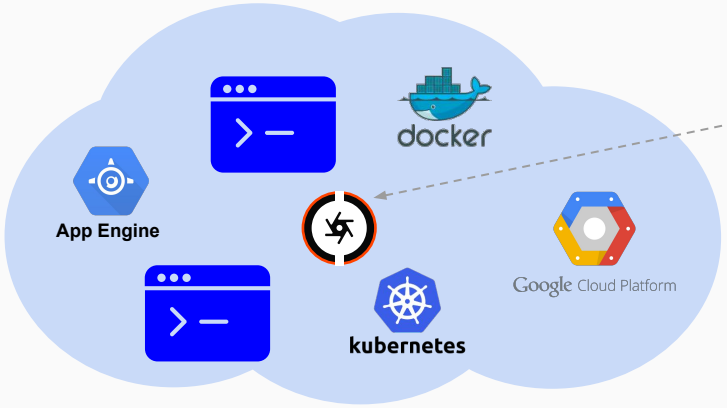


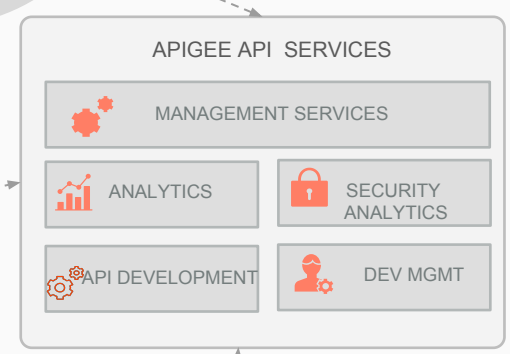
Apigee Microgateway

AWS Integration
Deploy Apigee Microgateway to **EC2 Container Service** or **AMI**

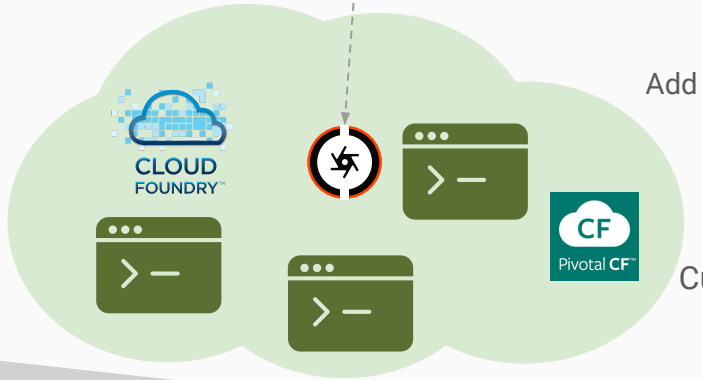


GCP K8S & App Engine Integration
Microgateway can be deployed in **K8S/GKE** or **App Engine**





Native CF Integration
Add Microgateway as a CF **Route service** or a **meta-buildpack**

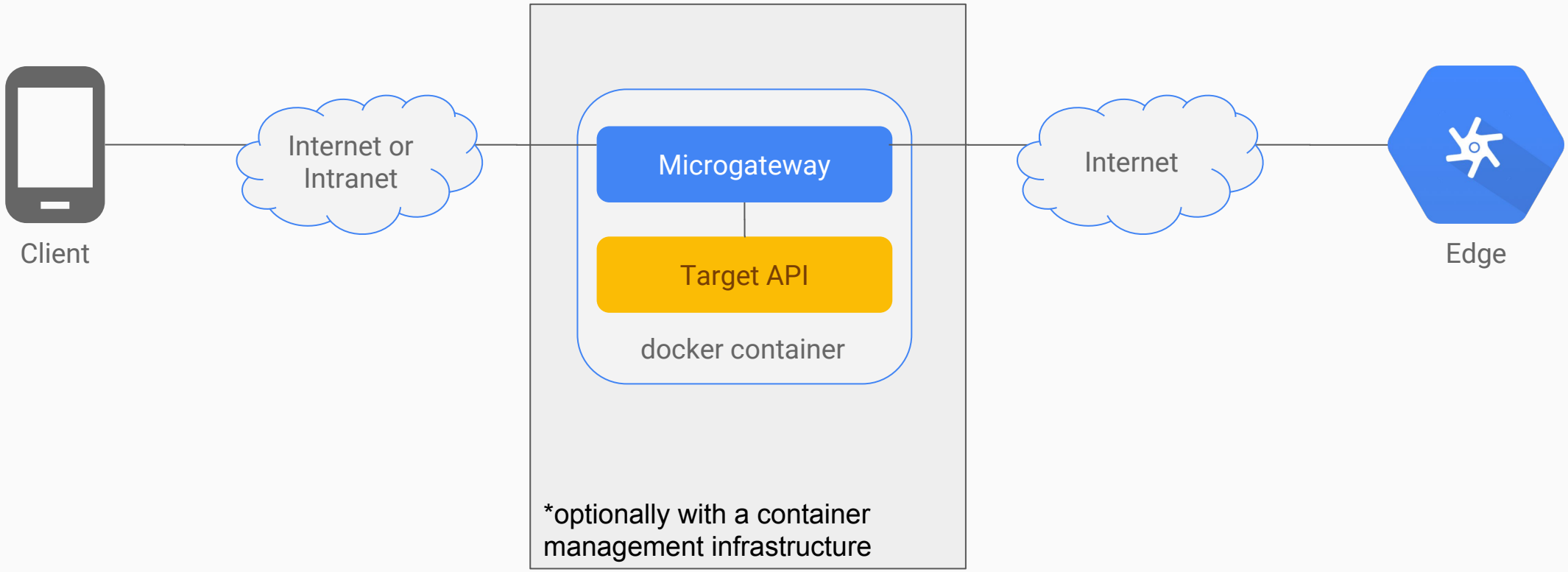


On-Premises or Customer Managed

Google apigee

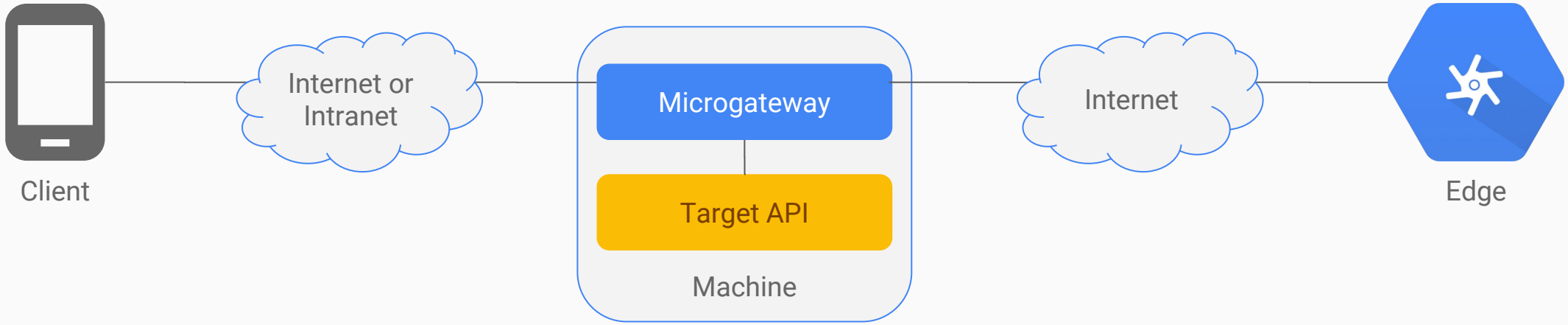
Deployment scenarios

Simplest deployment: microgateway and target in same container



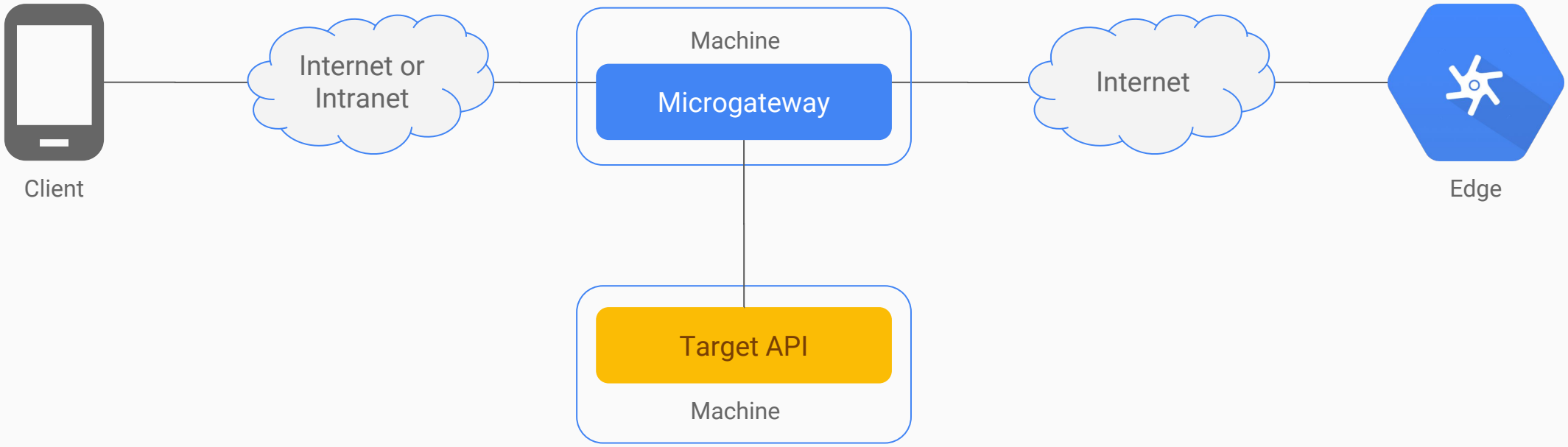
Deployment scenarios

Simplest deployment: microgateway and target in same machine



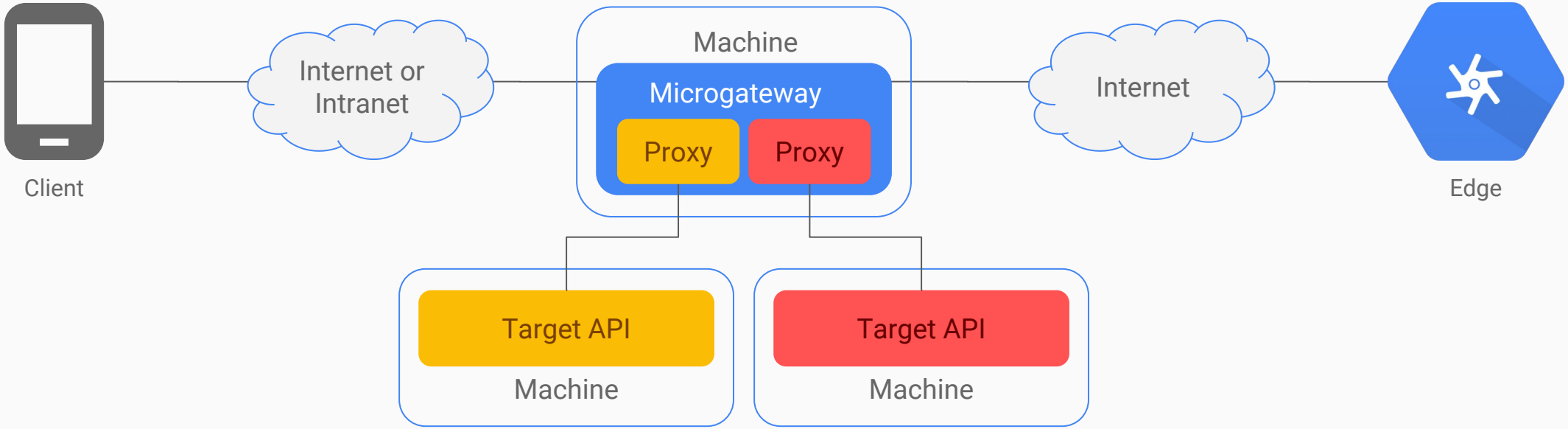
Deployment scenarios

Microgateway and target in different machines



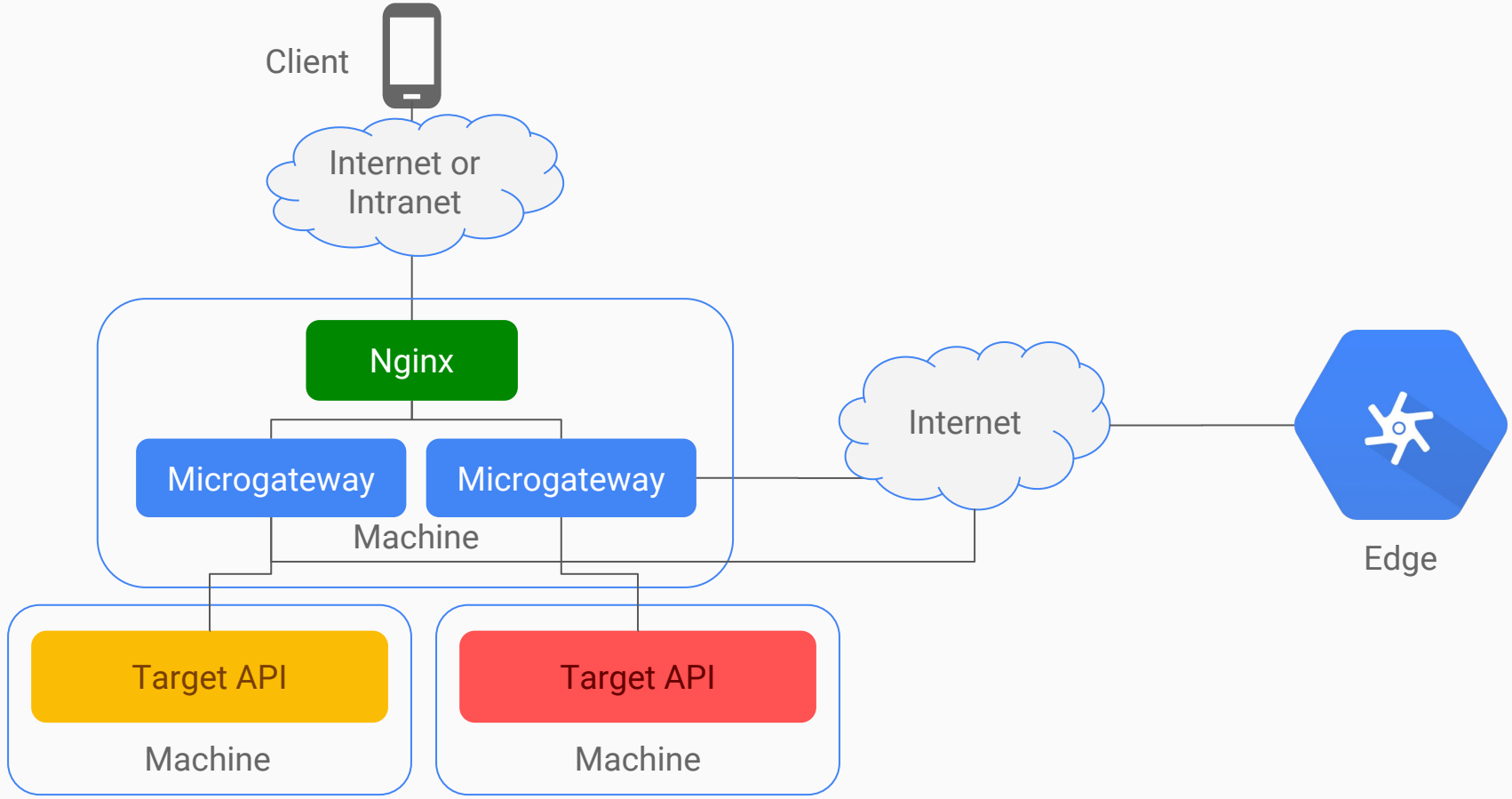
Deployment scenarios

Microgateway proxying multiple targets in different machines



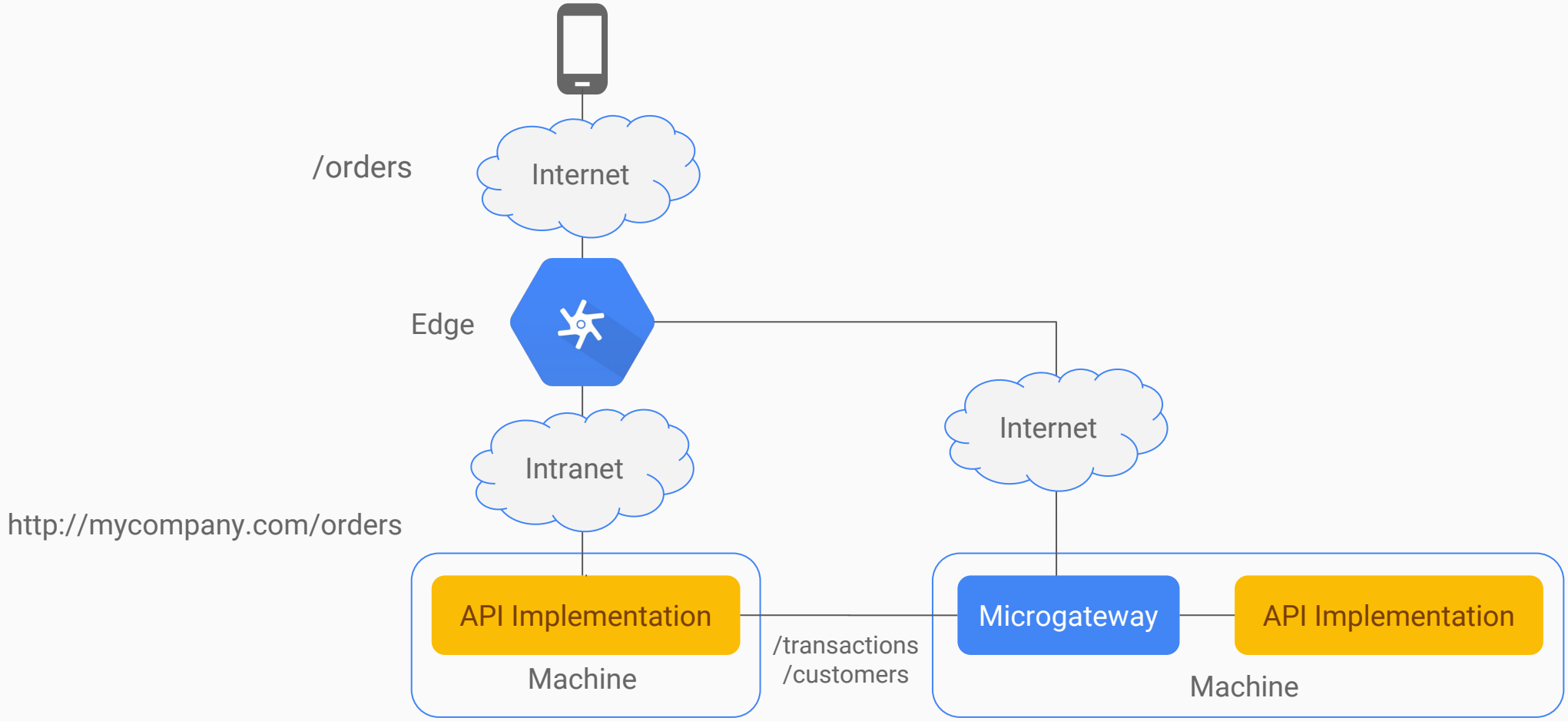
Deployment scenarios

Load-balancing and SSL termination



Deployment scenarios

Intranet traffic protection



Deployment

Microgateway

01

Prerequisites

- Node.js version 4.5 LTS or later installed.
- Install `edgemicro` and custom plugin modules globally:

  ```
  $ ~> npm install -g edgemicro plugin-1
  plugin-2 … plugin-n
  ```

  When using `npm` with `-g` option, the module is installed in `[prefix]/lib/node_modules` directory. The value of `[prefix]` can be obtained with the following command:

  ```
  $ ~> npm config get prefix
  ```

- Create a soft link in the plugins directory (`[prefix]/lib/node_modules/edgemicro/plugins`) for all your custom plugins

  ```
  $ plugins> ln -s ../../plugin-x .
  ```

02

Microgateway configuration

- Run the command below to generate the default configuration file:

  ```
  $ edgemicro init
  ```

- Run the `edgemicro configure` command with the following options:

  ```
  $ edgemicro configure -o <org> \
  -e <env> -u <email>
  ```

  When this command is run a new proxy called `edgemicro-auth` is created in the organization.

- Copy the key and the secret written to the standard output by the previous command, they are required to start the microgateway.
- Verify the installation:

  ```
  $ edgemicro verify -o <org> \
  -e <env> -k <key> -s <secret>
  ```

03

Entities creation

- Create a microgateway-aware proxy (name prepended with `edgemicro_`), with the desired base path and target urls.
- Create a product. Do not forget to add the `edgemicro-auth` proxy and the microgateway-aware proxy to the product and configure the quota attributes if required.
- Create a developer.
- Create a new developer app and assign the product and developer created. Keep the consumer key and consumer secret.

Deployment
Microgateway

04

API security configuration

- The `oauth` plugin should appear and lock as follows in the configuration file (`~/.edgemicro/<org>-<env>-config.yaml`) :

```
oauth:
    allowNoAuthorization: false
    allowInvalidAuthorization: false
```

- Verify that the `oauth` plugin is listed in the `plugins` section:

```
plugins:
    sequence:
        - oauth
```

Token creation and API key verification is done by the `edgemicro-auth` proxy. If another service wants to be used, it needs to be specified in the configuration file:

```
edgemicro:
    authUri: <uri>
```

05

Spike Arrest configuration

- Add the `spikearrest` plugin in the configuration file as a top-level element:

```
spikearrest:
    timeUnit: second
    allow: 10
    bufferSize: 10
```

- Configure the plugin attributes:

timeUnit (second|minute): window size units.

allow: Max. number of requests allowed in time unit.

bufferSize (optional, default = 0): Number of requests stored in a buffer. As soon as the next execution window occurs, buffered requests are processed first.

- Make sure that the plugin is listed in the `plugins` section:

```
plugins:
    sequence:
        - oauth
        - spikearrest
```

06

Quota configuration

If a quota was set for the product, add the `quota` plugin is listed in the `plugins` section:

```
plugins:
    sequence:
        - oauth
        - spikearrest
        - quota
```

The quota plugin requires the oauth plugin to be executed beforehand.

Deployment
Microgateway

07

Custom Plugins Configuration

- If the custom plugin has some parameters to configure, add a top-level element in the configuration file:

```
plugin-x:
    param-1:
    param-2:
```

- Add the custom plugin to the `plugins` list:

```
plugins:
    sequence:
        - oauth
        - spikearrest
        - quota
        - plugin-x
```

08

Server SSL Configuration

Add the edgemicro:ssl attribute to the configuration file.

```
edgemicro:
    ssl:
        key: <absolute path to key file>
        cert: <absolute path to cert file>
```

09

Client SSL/TLS Configuration

Microgateway can be configured as SSL/TLS client when acting as client. Settings can be for all/one host:

```
targets:
    (ssl|tls): # One or the other
        client:
            # Here go the properties
```

```
targets:
    host: <domain-name>
    (ssl|tls): # One or the other
        client:
            # Here go the properties
```

Properties:

pfx: path to file with client key and cert (PFX format).
key: path to file with client key file (PEM format).
passphrase: passphrase for PFX or private key file.
cert: path to file with client cert file (PEM forma)t.
ca: path to file with trusted certs (PEM format).
ciphers: supported ciphers separated by ':'.
rejectUnauthorized: does server cert need to be trusted?
secureProtocol: SSL method to use (eg: SSLv3_method)
servername: server name for SNI TLS extension.

Deployment
Microgateway

10

Logging Configuration

Logs can be sent to the `stdout` or to a log file:

```
edgemicro:
    logging:
        level: info
        to_console: true
```

```
edgemicro:
    logging:
        level: info
        dir: /var/tmp
        stats_log_interval: 60
        rotate_interval: 24
```

Properties:

stats_log_interval (seconds, default: 60): interval when the stats record is written to the log file.

rotate_interval(hours, default: 24): interval when log files are rotated.

11

Start Microgateway

Start the microgateway using the following command.

```
$ edgemicro start -o <org> -e <env> -k <key> \
-s <secret>
```

Alternatively you can set the following environment values so you don't need to specify the options in command line:

- `EDGEMICRO_ORG`
- `EDGEMICRO_ENV`
- `EDGEMICRO_KEY`
- `EDGEMICRO_SECRET`

If changes are done to the configuration file once the microgateway is started you can use `edgemicro reload` command so they are applied without restarting. That command takes the same options as `edgemicro start` command.

12

Verify API Security

You can make calls to microgateway using the API key or access token.

API Key

```
$ curl http://localhost:<port>/<path> \
-H "x-api-key: <consumer-key>"
```

Access Token

```
$ edgemicro token get -o <org> -e <env> \
-i <key> -s <secret>
```

```
$ curl -H 'Authorization: Bearer <token>'
http://localhost:<port>/<path>
```

Deployment
Microgateway

In case microgateway needs to be started as a service, `pm2` could be used; you just need to follow the steps below:

1. Install pm2:

   ```
   $ npm install -g pm2
   ```

2. Start edgemicro using `pm2`:

   ```
   $ pm2 start edgemicro -- start -o <org> -e <env> \
   -k <key> -s <secret>
   ```

   Starting `edgemicro` like this has two advantages: it will be restarted if it crashes and unhandled exceptions would be recorded in a log file (`$PM2_HOME/logs/app-err.log`, By default `$PM2_HOME` is `~/.pm2`).

3. Check whether pm2 has started properly:

   ```
   $ pm2 logs edgemicro
   ```

4. Additional information about the process running (pid, uptime, cpu usage, memory usage,...) can be obtain with this other command:

   ```
   $ pm2 list
   ```

5. Get the automatically-configured startup script for your machine:

   ```
   $ pm2 startup
   ```

   It automatically detects the available init system, but you can specify it yourself to (eg: `pm2 startup systemd`). You can also use option `-u` to specify the user and `--hp` to specify the home directory.

6. Once the scripts are generated keep a list of the processes running, so they will be started at boot time:

   ```
   $ pm2 save
   ```

7. Reboot the machine and check that microgateway is running.

8. Smile :).

Deployment

Private `npm` repository

Set up a private `npm` repository server to install `edgemicro` node module and its dependencies, as well as the node packages of the custom plugins being used. The advantages of doing this are:

Private modules

You get all the benefits of an `npm` package system without sending all code to the public, and use private packages (eg: custom microgateway plugins) as easily as the public ones.

Cache for npmjs.org registry

If you have more than one server to install modules on, this decreases latency and provides limited failover (if npmjs.org is down, you might still find something useful in the cache).

Override public packages

If a modified version of some third-party module is required, it can be published locally under the same name.

Deployment

Private `npm` repository: Sinopia

FEATURES

- Local npm registry with minimal configuration.
- No need to install and replicate an entire CouchDB database, as it has its own a database. If it doesn't have a package, it fetches it from npmjs.org, keeping only those packages used.

INSTALLATION INSTRUCTIONS

```
$ npm install -g sinopia forever
$ forever start `which sinopia`
```

When you start a server, it auto-creates a configuration file `config.yaml` that you can later modify.

USER CREATION

```
$ npm adduser --registry http://localhost:4873
```

NPM CLIENTS CONFIGURATION

```
$ npm set registry http://<sinopia-server>:4873
```

Deployment

Private npm repository: Sinopia

SINOPIA CONFIGURATION FILE

```
# path to a directory with all the packages
storage: ./storage

auth:
  htpasswd:
    file: ./htpasswd

npmjs:
  url: https://registry.npmjs.org/

packages:
  # scoped packages
  '@*/*':
     # keywords: $all, $anonymous, $authenticated
      access: $all
      publish: $authenticated
     proxy: npmjs
  '*':
      access: $all
      publish: $authenticated
      proxy: npmjs

logs:
  - {type: stdout, format: pretty, level: http}

listen:
  - 0.0.0.0:4873
```

Deployment

NGINX

LOAD BALANCER (SAME MACHINE)

1. Install `nginx`.
2. Edit the configuration file `/etc/nginx/nginx.conf` and add the following in the `http {}` block:

```
upstream example-servers {
    server localhost:8001;
    server localhost:8002;
}
server {
    listen       80;
    server_name  emgw;

    location /<path> {
        proxy_pass http://example-servers;
    }
}
```

3. Start the service

Deployment

NGINX

LOAD BALANCER (DIFFERENT MACHINES)

1. Install `nginx`.
2. Edit the configuration file `/etc/nginx/nginx.conf` and add the following in the `http {}` block:

```
upstream example-servers {
    server <mgw1-ip>:8000;
    server <mgw2-ip>:8000;
}
server {
    listen       80;
    server_name  emgw;

    location /<path> {
        proxy_pass http://example-servers;
    }
}
```

3. Start the service.

Deployment

NGINX

LEAST-CONNECTED

Determines which upstream server has the least amount of outstanding connections alive and passes more traffic to it.

```
upstream example-servers {
    least_conn;
    server <mgw1-ip>:8000;
    server <mgw2-ip>:8000;
}
```

IP HASH

Maps the client IP to a server in the list, always sending requests from a single client to the same server.

```
upstream example-servers {
    ip_hash;
    server <mgw1-ip>:8000;
    server <mgw2-ip>:8000;
}
```

WEIGHTED

```
upstream example-servers {
    server <mgw1-ip>:8000 weight=3;
    server <mgw2-ip>:8000;
    server <mgw3-ip>:8000;
}
```

Bootstrap process

01

Download public key for JWT validation

02

Download list of microgateway-aware proxies

03

Download list of API products and their attributes

Plugin authoring

1. Create a new directory and change to it.

```
$ ~> mkdir <plugin-name>
$ ~> cd <plugin-name>
```

2. Initialize a node project:

```
$ ~> npm init
```

3. Create a file named `index.js`, copy the contents below and save it.

```
'use strict';

var debug = require('debug')('plugin:plugin-name');

module.exports.init = function(config, logger, stats) {
    return {
        onrequest: function(req, res, next) {
            debug('onrequest');
            next();
        },
        onend_request: function(req, res, data, next) {
            debug('onend_request')
            next(null, data);
        },
        onend_response: function(req, res, data, next) {
            debug('onend_response');
            next(null, null);
        }}
};
```

4. Once the plugin is developed, publish the module to your private npm repository.

```
$ ~> npm set registry http://<sinopia-server>:4873
$ ~> npm login
$ ~> npm publish
```

5. In the machine where the microgateway is installed, install the plugin as a global node module from your private repository.

```
$ ~> npm set registry http://<sinopia-server>:4873
$ ~> npm install -g <plugin-name>
```

6. Create a soft link in the edgemicro plugins directory to the custom plugin.

```
$ plugins> ln -s ../../<plugin-name> .
```

7. Add it to the list of plugins in the microgateway configuration file.

```
plugins:
    sequence:
        - oauth
        - spikearrest
        - plugin-name
```

Plugin authoring

Request handlers

onrequest
ondata_request
onend_request
onclose_request
onerror_request

onrequest

Fires when the first byte of the request is sent. Gives access to the headers, URL, query parameters, and HTTP method.

ondata_request

Called when a chunk of data is sent. Passes the data to the next plugin in the sequence. The data returned by the last plugin is sent to the target.

onclose_request

Indicates that the client connection has closed.

onend_request

Called when all of the request data has been sent from the client.

onerror_request

Called if there is an error sending the request from the client.

Plugin authoring

Response handlers

onresponse
ondata_response
onend_response
onclose_response
onerror_response

onresponse

Fires when the first byte of the response is received. Gives access to the response headers and status code.

ondata_response

Called when a chunk of data is received from the target. The payload can be transformed here.

onclose_response

Called when the socket connection to the target is closed.

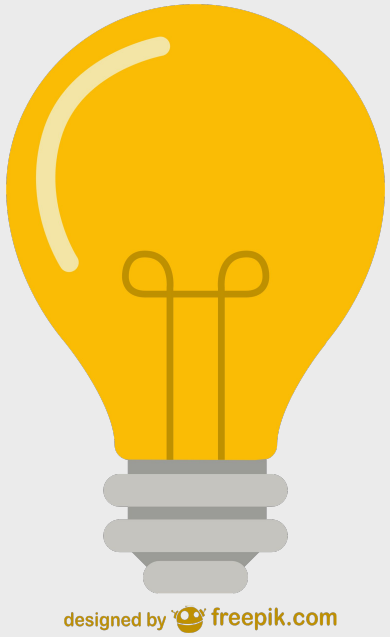
onend_response

Called when all of the response data has been received from the target. The payload can be transformed here too.

onerror_response

Called if there is an error receiving the target response.

Plugin authoring



Things to remember when writing a plugin:

- Request / response handlers must call **next()** when done processing. If not, processing will stop and the request will hang.
- The first argument to **next()** may be an error which will cause processing to terminate.
- The **ondata_** and **onend_** handlers must call **next()** with a second argument containing the data to be passed to the target or the client. It can be null if the plugin is buffering and has not enough data to transform at the moment.
- A single instance of the plugin is used to service all requests and responses. If per-request state needs to be retained between handler calls, it can be saved in a property added to the supplied **request** object, whose lifetime is the duration of the API call.
- Catch all errors and call **next()** with the error. If not, the API call will hang.
- Do not perform compute-intensive tasks in the main thread as this can adversely affect performance.

Plugin authoring

URL rewriting

```
onrequest: function(req, res, next) {
    var baseUrl = res.proxy.parsedUrl.pathname;
    var proxyBasepath = res.proxy.base_path;
    if(proxyBasePath === '/users') {
        req.targetPath = baseUrl + '/customers'
    } else if(proxyBasepath === '/businesses) {
        req.targetPath = baseUrl + '/compies'
    }
    //…
    next();
}
```

Payload transformation

REQUEST

```
onend_request: function(req, res, data, next) {
    var payload = JSON.parse(data,toString());
    delete payload.password;
    next(null, JSON.stringify(payload));
}
```

RESPONSE

```
onend_response: function(req, res, data, next) {
    var payload = JSON.parse(data,toString());
    delete payload.password;
    next(null, JSON.stringify(payload));
}
```

Custom logging

If the built-in logger does not fit with your specific logging requirements, a custom plugin can be created. Here's an example of a Bunyan JSON logger:

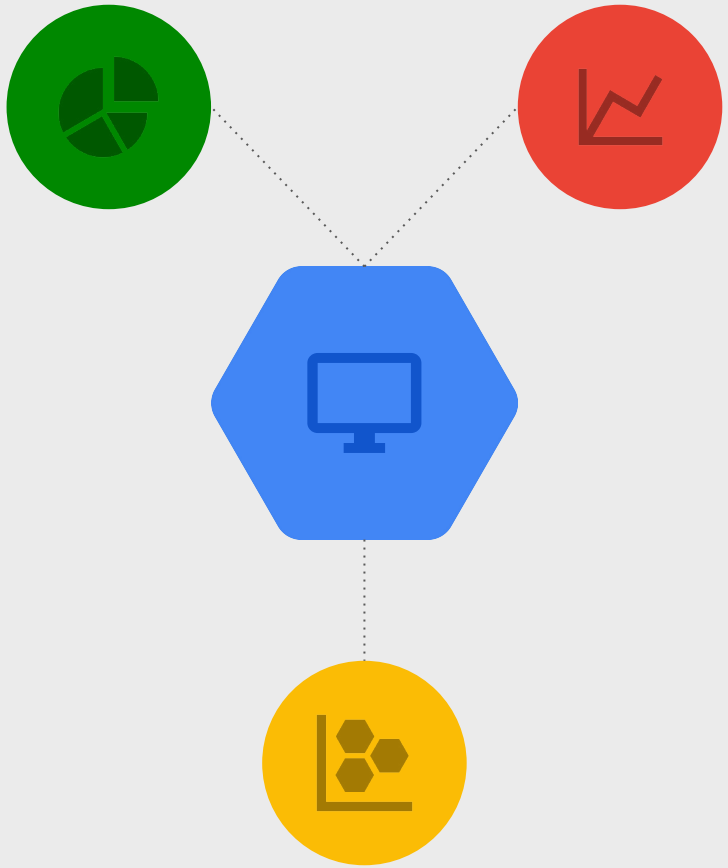
```
 var bunyan = require('bunyan');

module.exports.init = function() {

    var log = bunyan.createLogger({
        name: "bunyan-logger-plugin",
        streams: [ { level: 'error', stream: process.stdout }, { level: 'info', path: '/var/tmp/mgw-bunyan-info.log' }]
    });

    return {
                onrequest: function(req, res, next) {
                    log.info('ONREQUEST');
                    next();
                },
                onend_request: function(req, res, data, next) {
                    log.info("ONEND_REQUEST");
                    next(null, data);
                }
            };
}
```

Monitoring



If `edgemicro` has been started using `pm2`, monitoring the CPU and memory usage is simple:

```
$ pm2 monit
```

Integration with Keymetrics monitoring dashboard is quite easy:

1. Sign in for a Keymetrics account
2. Create a new bucket
3. Run the following command with the provided secret and public keys:

```
$ pm2 link <secret-key> <public-key>
```

Thank you