eMobility – TAM diagram for Threat Modeling



# Use-cases

## User login:

* User opens his web browser or mobile app
  + From browser: user visits URL, prefix is her tenant
  + From mobile: user selects tenant
* Static pages are retrieved from ev-dashboard, login page displayed
* User provides login/password via REST call to ev-rest-server
* Ev-rest-server looks-up user in ev-mongodb, returns JWT token in case of success
* Further interactions take place via REST calls between user and ev-rest-server passing the token

## User onboarding:

* At login screen, user visits Register page
* User provides credentials (email, password, CAPTCHA) via REST call to ev-rest-server
* Server creates inactive user in ev-mongodb
* Server sends email via SMTP to Ionos mail server with activation token
* User visits link provided in email to ev-rest-server
* Ev-rest-server matches token and activates user
* If configured so, server auto-assigns user authorization to see and use charging stations
  + If not, manual action is necessary for a tenant admin

## Lost password:

* At login screen, user visits Lost Password page
* User provides email address and solved CAPTCHA via REST call to ev-rest-server
* Server sends a reset password email with an activation token
* User visits link (call to ev-rest-server) with token
* User receives an email with her new password (which can be changed from the profile page)

## Charging station onboarding:

* An operator decides what the charging station ID should be and which tenant it should connect to
* Operator connects to the charging station directly, switch to ‘supervised mode’ and configures the URL (including tenant ID and station ID) of the ev-json or ev-soap server
* The charging station registers itself via OCPP protocol to the relevant server and is now operational

## User badge onboarding (optional):

* User badges on a charging station
* Charging station sends authentication request to its server
* Server queries ev-mongodb for a badge match
* Since the user does not exist, the server will create a new Unknown user with this BadgeID
* A tenant administrator manually updates the badge list of the user with the badge ID found in the new Unknown User profile
* Tenant administrator deletes the Unknown User

## Start session with Badge:

* User badges on a charging station
* Charging station sends authentication request to its server
* If the corresponding Organization is not active (Site, Site Area) and the User is active, the request succeeds
* Otherwise, the server checks if the Charger is assigned to a Site Area and if the User owning the badge is assigned to a Site and this Site owns the Site Area above
* Finally, it checks that the user is Active
* If all the tests pass, the charging station is started

## Start session with mobile app:

* User logs into his mobile app selecting the Tenant and got the JWT token
* User makes a REST request to ev-rest-server to start an available station with his virtual badge in his JWT token
* Ev-rest-server searches for the station in the database
  + If the station is connected via SOAP, ev-rest-server sends a start session request to the ev-soap-server who will dispatch the call to the proper charging station
  + If the station is connected via web socket, ev-rest-server searches for the right ev-json-server and asks it to start the session
* Charge station server initiates the start session packet exchange for this badge id (Authorize, Start Transaction, …) with the charging station via OCPP calls

## Stop session with Badge:

* User badges on a charging station
* If badge id maps one of the badge ids used for starting a session, the charging station sends authentication request to its server (otherwise the charger drops the request)
* Charge station stops the session

## Stop session with mobile app:

* User logs into his mobile app
* User makes a REST request to ev-rest-server to stop a running session with his virtual badge
* If the user is authorized, the ev-rest-server makes a stop session request to the relevant charge station server as described in the Start session with mobile app use-case
* Charge station server checks that the virtual badge id matches with the one which was used to start the session. If successful, it initiates the stop session packet exchange (Authorize, Stop Transaction, …) with the charging station via OCPP calls

## Stop session with car:

* User opens his car
* The charging station stops the transaction

## Analytics:

* Setup use-case:
  + User logs into SAP Analytics server
  + Through this server, user connects with his username/password to the Analytics oData server
  + Credentials are checked against ev-rest-server
  + If user role permits (currently: admin, demo), data the user is authorized to pull is sent to analytics server through oData (once a day)
* Data consumption:
  + If role permits, user can click a link from his web session
  + User is redirected to the appropriate dashboard of SAP Analytics

## OCPI:

* OCPI module is configured with Gireve’s URL and key
* Gireve accepts (trusts) data sent from OCPI service (currently: information about station availability, planned: roaming user authentication requests and electricity costs to apply)

## Billing:

* Ev-rest-server configured with username/password and URL of the Covenant Charging service
* Ev-json-server/Ev-soap-server push events: session start, stop, and mid-point ranges

## Expense management:

* From his browser, user asks to connect to Concur
* Ev-rest-server triggers a request to Concur
* Concur runs an OAuth2 registration process
* Upon success, the OAuth2 token is stored in the user table
* User can now create expenses from his browser to the Concur server for his past sessions

## Tenant creation:

* Pre-configured superadmin user or superadmin user created by another superadmin user logs into master tenant (main domain URL)
* Superadmin creates a new tenant (with a default administrator user) and set it up (activate components like Refunding, Analytics, Organization…)
* The ev-rest-server triggers the generation of relevant mongodb collections on their own schema and create an admin user in this tenant that will receive an activation email

## Tenant admin:

* Administrator logs in through his browser or mobile app
* Administrator add/removes/locks/unlocks users, change their role
* Administrator start/stop charge sessions
* Administrator deletes charging stations (revoke)

## Code push :

* User launches cloud foundry command-line interface
* User logs-in against SAP IDP with username/password with cf login
* User sends push command
* Code is compiled, obfuscated, minified then actually pushed to the production server

# Services

## Mongodb

* Requested on the SCP space page
* Creates a mongodb instance together with access credentials (username / password)
* These credentials are sent to linked containers automatically upon deployment through environment variables
* Is used by ev-rest-server, ev-json-server, ev-soap-server, ocpi

## Application logs

* Requested on the SCP space page
* Let application store data in a specific tenant, configured with an API key
* API key is sent to linked containers automatically upon deployment through environment variables
* Can be consumed from an SCP Kibana service
* Is used by ev-rest-server, ev-json-server, ev-soap-server, ev-ocpi-server, ev-odata-server

## Dynatrace

* Specified as dependency for servers
* Cloud foundry automatically fetches and deploys agents where necessary
* Deploys a Dynatrace server SCP users can connect to
* Is used by ev-rest-server, ev-json-server, ev-soap-server, ocpi, Analytics oData

# Assets

Mongodb tables contain schemas (collection prefix) defining the tenants. Master tenant is called ‘default’. Each tenant contains a user table containing user email, hashed password. It will also contain email confirmation links for user activation and for lost password requests, badge id(s), virtual badge id, and OAuth tokens to connect to Concur.

Upon successful authentication, users receive a signed JWT token (including username, role, badge id…) valid for 12h which get stored inside the browser’s local storage space.

Each server contains an asset called config.json where different credentials are stored, such as the key to encrypt JWT tokens, the key to encrypt sensitive data (application-level message-based encryption) the CAPTCHA secret key, credentials to the email server, the API key to communicate with Gireve, etc.

# Application Roles

Superadmin: can add/remove Superadmins, can create/delete tenants, can check the Logs, can only access the Super Admin tenant

Demo: can see free/occupied stations, see the anonymized statistics, the anonymized sessions, the organization (company, site, site area)

Basic: Demo, plus: can edit/delete own user data, can start/stop own session on charging station (with proper badge ID or virtual badge ID), can see own charging sessions, own statistics

Admin: Basic, plus: can add/edit/remove users, can delete charging stations, can start/stop all sessions, can see all charging sessions, can see stats of all users

Site Admin: is a Basic user + an Admin user but only for a limited list of Sites