**Apigee Training – Developer Training**

January 11, 2016

Steven Richardson

Terry David

**Day 1**

Kohl’s open api

Digital Insights

TCS/Amex

Accenture

TSI

Platform Basics/Introduction

* Security across the value chain
  + Data driven API protection
  + Omni channel identity
  + Developer-centric security
  + Threat protection & governance
  + Plugable identity (backend as a service has IAM)
* BaaS
  + Wrapper around “Usergrid”
  + Enterprise level services on top of Usergrid project
  + NoSQL database API driven
  + Uses OAuth does not support OpenConnect Id
  + Services
    - Datastore
    - User data
    - Connections/social? (pre-built integration to FB/Twitter may be)
    - Location queries
    - Push notifications
* Apigee 127
  + Node.js application, Swagger 2.0 API specification
* Developer Portal/Smart Docs
  + Based off Drupal
  + Does not have ability to provide API level access control on docs (meaning show API documentation only if the application/developer has access to it)
* Apigee console
  + Apigee’s REST client
  + Resides on public cloud, so isn’t popular for on-premise integrations
* Monetization Capabilities
  + Revenue-sharing models
  + Fee-based models
  + Freemium models
    - Duration
    - Quantity
    - Hybrid
  + Has a separate PCI compliant add-on (paid) that will make the monetization pack PCI compliant
* API Analytics
  + Error Report
  + Custom reports
  + Has ability to push data to Splunk server (Syslog policy)
  + Syslog policy has changed a lot over time – added TCP, SSL support, async call
* BaaS Capability
  + You can make changes (key=value) which can do a server side injection to all mobile apps (change value globally in the mobile).
  + E.g. change the server address without rolling out an update
* Apigee Insights
  + Apigee’s Big Data solution
* Platform Behind the Scenes (components of the Apigee platform)
  + Base setup ~2k tps (simple proxies)
  + Apigee does not follow “connector based models” – you can build/maintain it yourself using the node.js package (volos) (npm library)
  + Components (node based – horizontally scalable)
    - Load Balancer
    - Gateway (run-time)
      * Router
        + If you are terminating SSL here, you might need to scale this
      * Message processors
    - Analytics backend (
      * Qpiid
      * PostgreSQL
    - BaaS
    - Developer Portal
    - Management Server
    - Apigee UIs
    - Infrastructure services – non-apigee (zookeeper, qpid, Cassandra) (run time)
* Components
  + PostgreSQL for reporting/analysis
  + Zookeeper (configuration metadata store)
  + Cassandra (NoSQL database )
  + Qpid for queuing
* Logical divisions
  + Planets (~equivalent to Data center)
  + Pods
  + Organization
  + Environment

**Anatomy of API Proxy**

* Organization
  + Has environment
  + Logically separate
  + Has users, roles, applications, keys, resources and vaults
* Flows
  + Global flows
    - Every request will run these policies
  + Conditional flows
    - Drives analytics and separates out logical
    - Order is important, runs top down
    - Only 1 conditional flow will be executed
  + Post Client flow
    - Earlier used to send data to syslog after sending the response.
    - Policy got in more efficient, (sends syslog in async) so no longer necessary to use the Post Client flow
* Route Rules
* Target Endpoints
  + Load Balancing policy
    - Allows you to emulate load balancer for backend
    - Multiple rules available (RR, least connection, weighted)
    - Under HTTPTargetConnection block
* Health Monitor
  + Making backend calls to ensure target is working/up and running.
* Fault rules
  + Individual fault rules applied based on a condition that has occurred
  + Order dependent – gets processed bottom to top
  + Trigger
    - Internal situation which causes a fault (within edge – JS error)
    - Faults from target & callouts
    - Manual/programmatic trigger
* Advanced Endpoint Properties for Proxy
  + Streaming policies states, do not put anything in the message buffer as I will not be looking into the message.
  + If this is enabled any policy that looks into the message payload does not fire\*\*\*
  + Connect.timeout.millis and io.timeout.millis to setup
  + Success.codes – used to trick the system to say these are the codes for which it would go in the normal flow and not trigger the fault rules. By default, its value are 1XX, 2XX and 3XX.

Connecting Your API

* New API Proxy
  + Import WSDL – not recommended – use it as backend service with SOAP as the backend.
  + Most popular are “Backend Service” and “No Target”
* Fault Rules
  + Are not default filled up by UI – has to be hand written
  + The order is below to up
* Key Value Map Policy
  + Initial value
    - Never used, take it out
  + Put and Delete
    - Should never happen in run time. So if it is used it is a red flag
  + Scope
    - Should be “environment” or “organization” based on where the KVM has been created \*\*\*\*
  + Get
    - Key 🡪 Parameter
      * The documentation is basically a reference to a reference. Instead of using the “reference” attribute, use it between tags
      * <Parameter>kvm-key</Parameter>
* Basic Authentication
  + Same policy can be used to decode BA header ☺
  + Assign to, createNew should be set to “True” as keeping it “false” will not create new headers if incoming requests have headers (relevant during oAuth)

**Day 2**

API Product Strategies

* Common security and Authorization Patterns
* API product in Apigee is a grouping of API resources
  + Can come from one or more APIs
  + Application developers register themselves to gain access to the product
* API Product Strategies (common)
  + ApiProxy Model
    - Centered around creating a 1:1 relationship with your API proxies and your products
    - Access model has to be associated with multiple products – which product wins scenarios in terms of quotas, etc.
    - Becomes complex to manage as you grow
  + Service Plan Model
    - Tiered access system
  + Business Model
    - Define products by business units
  + Ownership Model
    - Based on API development ownership
* Product is tied to access and quota in Apigee by default \*\*\*\* (Again they are saying that is not the only way, there are ways around)
* Key concepts
  + Product to application is many to many relationship
  + When a product is associated with an Application a Key is generated
  + Products contain configuration used to display information and control workflow in the Developer Portal
  + Products can be made “internal”, “private” and “public”
  + You can set custom configuration that can be accessed with every API calls to the product
  + The information that you enter in the Create product page will be shown on the public facing Developer portal. \*\*\*
* Access Control
  + Not built-in by default to restrict access on HTTP verb
  + Access control by default takes the last registered product as the meta-data
  + What I am hearing from Apigee is that there are ways to work-around these, but ….
* Extending API Products
  + Products can be extended via custom attributes
  + Variable flow.resource.name allows you to validate based on other information i.e. basepath, verb
    - Use it in the pre-flow
    - Normally used with a JS policy
    - You can use this to custom make your path and then check
    - Workaround for restricting access on verb (\*\*\*)
    - Restriction: If your product is defined in an environment, using flow.resource.name does not take the environment into account????
    - Where and how is it used??
* Creating a Product
  + UI
* Creating a Developer
  + Custom attributes can be added to a developer
  + E.g. – set up developer level quota
    - And then at the flow level you can check the quota at different levels (quota hierarchy) and choose the appropriate level to provide the quota

**Threat Protection**

* Common Threat Patterns
  + Fuzzing
    - Someone is doing some probing – get some response back and based on the response work on the next steps
  + Malicious inputs
  + Injection
    - Sending embedded SQL
  + XSS
  + DDoS
* JSON Threat Protection Pattern
  + Can check the structural stuff with JSON payload before actually parsing the JSON
  + Less focus on it now, since earlier JSON Parsers were not good
* XML Threat Protection Pattern
  + Again checks the structure before parsing it
  + Content type has to be setup for this the policies
* Performance impact
  + There is a cost to running these policies, but depends on how big and many inputs coming in
* Sample Regular Expression
  + You have to URL encode and put it in the XML
  + There is a cost as regular expressions search in the payload
  + Note: RegEx can be written badly or fine tuned
  + RegEx can be stacked, but best practice for Apigee – do it in one. Would have been good
  + This is a black list in Apigee
  + Can use any regEx
  + Can be used to run a counter and have a quota based on number of bad request counts
* CORS support
  + Proxy creation on wizard
  + Additional updates required for OPTIONS

Securing Your API

* Authentication and Authorization
* API Key validation

Common Authentication Pattern

* API Key
* Access Token

OAuth Support

* Authorization server on its own
* Support grant-type = “Client credentials”
* The client credentials OAuth Policy is already built into the version that is created (when activating it). However other grant types, you will have to create APIs for those workflows (for developers to make a call to fetch Access tokens).
* Refresh token has setup for how many times can be refreshed. Once refresh token expires, you have to go through the entire loop again
* Strength of Apigee
  + You can associate custom data elements to the token \*\*\*\*
  + The user just sees the token, and when the token comes to you, you can see all the custom data that is associated with the token (e.g. account #)
* In order to fetch the values associated with the access token, there are ways to do it (slide #34) other than validation of the token (invalid tokens has to take this route)
* Slide #35, 36 – when trying to change some of the custom data elements attached to the token
* <http://docs.apigee.com/api-services/content/oauth-home>

Spike Arrest

* Spike Arrest Policy
  + Very light weight, adds 1-4 ms.
  + Rate is for each MP (Message Processor)
  + For implementation converts into interval of time between 2 calls (so does not do counts). It just looks at current request time stamp and last request time stamp.
  + Spike arrests at the API proxy level, not at a consumer levels

Quotas

* Different types
  + Calendar
  + Rolling Window
  + Flexi (mostly used on a monetization project)
    - One-time quota, has to be manually refreshed after flexi period is up

Distributed Quotas

* Important attributes
  + <Distributed> = True, bucket is shared amongst all MPs
    - Makes it very expensive, especially when multiple data centers
  + <Synchronous>, specifies whether the distributed quota counter is updated synchronously
    - Makes it very expensive, adds up to 100 ms latency
  + <AsynchronousConfiguration>, specifies how an aynsch distributed quota is updated. Default is once in 5 secs and batch of 5
  + If <PreciseAtSecondsLevel> is set to true, the quota will be enforced with an accuracy of a second, even if the <TimeUnit> is set at a unit longer than a second
    - Breaks up the quota to a second level and maintains alignment at that
* Best way to achieve high accuracy with a low latency is
  + <Distributed> is true and <Synch> as false

Flow Variables, gets populated to after the policy is run. You can use flow variable values after the policy is run for different purposes

target.copy.pathsuffix is the variable that needs to be set to false if you do not wish the path suffix to be copied from request to target

Performance Testing in Cloud (ELB)

* If the 2 server instances are in 2 Availability zones, ELBs sends requests from single source to a single Availability zone – hence will not work.

Using OAuth Validation

* Deploy OAuth Policy to an environment
* Make a call to OAuth API endpoint (in this case /oauth/client\_credentials/accesstoken) with the following
  + Add grant\_type=client\_credentials as a query parameter
  + Add API Key and API secret as Basic Auth credentials
* This will send back access token
* Once you get the access key, you next need to make a call to the actual API by sending the access token as a Bearer token in “Authorization” header
  + Authorization Bearer <access token>

Day 3

Apigee & Node.js

* Apigee Edge Vs Node.Js
  + There are overlap of features between Edge and Node.js
    - Examples are Spike Arrest, oAuth (Volos – package developed by Apigee and openly available)
  + There are areas where node js is good – e.g. SAML
* Which one to choose – hybrid approach
  + Edge configuration
    - Advantages
      * Performance
      * Ease of use
      * Governance
    - Disadvantages
      * Hosted development platform (team development is difficult is what they are saying)
      * Flexibility?
      * Sync flow processing \*\*\* (if you have APIs which are making out several call outs and does things in parallel, use Node.js)
  + Node JS coding
    - Advantages
      * Local development
      * Can be Platform agnostic (lot of framework for IoTs)
      * Can hand async processing
      * Uses Node skillset
      * Can support non-standard connection (node.js has built-in packages for 3rd party connectors)
    - Disadvantages
      * Performance can be an issue for high volume APIs (since runs on run-time *trireme* environment) (Apigee Java based run-time for node – under the hoods is Rhino heavily extended for Node.js)
  + Recommendation: combination of both
* Volos
  + Little difficult to work
  + Includes micro-governance – governance components within the API (built into the code rather than in externalized policies)
* Hosting Node.js on Edge
  + Slightly limited in terms of the local access that Node.js can do, as it is on hosted environment
  + Advantages with hosting on Edge platform – having access to everything on Edge platform, like flow variables, vault, etc. There is a apigee access package in Node.js built by Apigee
  + SSL support for node.js is not super. Difficult to support .pem files, but can support .jks files

Model Driven Design = Contract driven development

* Swagger
  + BDD (Business Driven Design) – using user language to build APIs
  + Editor Swagger 🡪 Generate Server 🡪 Node.js
  + Other Scaffolding tools
    - Yeoman
  + Apigee capabilities
    - Smart Docs can import Swagger specification for documentation
    - On the roadmap to add Edge proxy support for
* Using Swagger Editor for Node.js
  + New tags to add
    - x-swagger-router-controller at paths🡪/resource 🡪 next level
    - operationId at paths🡪/resource 🡪operation 🡪 next level
  + Change host to local host
  + Change basepath to “/” as it is taken care by Edge
* Node.js
  + Index.js is the entry point for your node.js module

var swaggerTools = require('swagger-tools'); is the one that allows using the Swagger specification

* Once you download the node.js server package from swagger, extract it.
* Run npm install in the folder that has index.js
  + This downloads the dependencies and packages in the node\_modules folder
* To run this application run node index.js (provided you do not have other application listening on the port marked in *serverport* variable in index.js file)
* nodemon is a better utility than node – it allows changes to files auto compiled and reflected as soon as they are saved. To install nodemon run

npm –g install nodemon (“-g” saying do a global install)

Best Practices

* Build your own folder structure – model, services
* Use of adapters (e.g. BaaS adapter) – more of a convenience object to talk to backend – utility folders

Sinon – mock library on Node.js (relatively complex, used when you are doing test driven development – TDD)

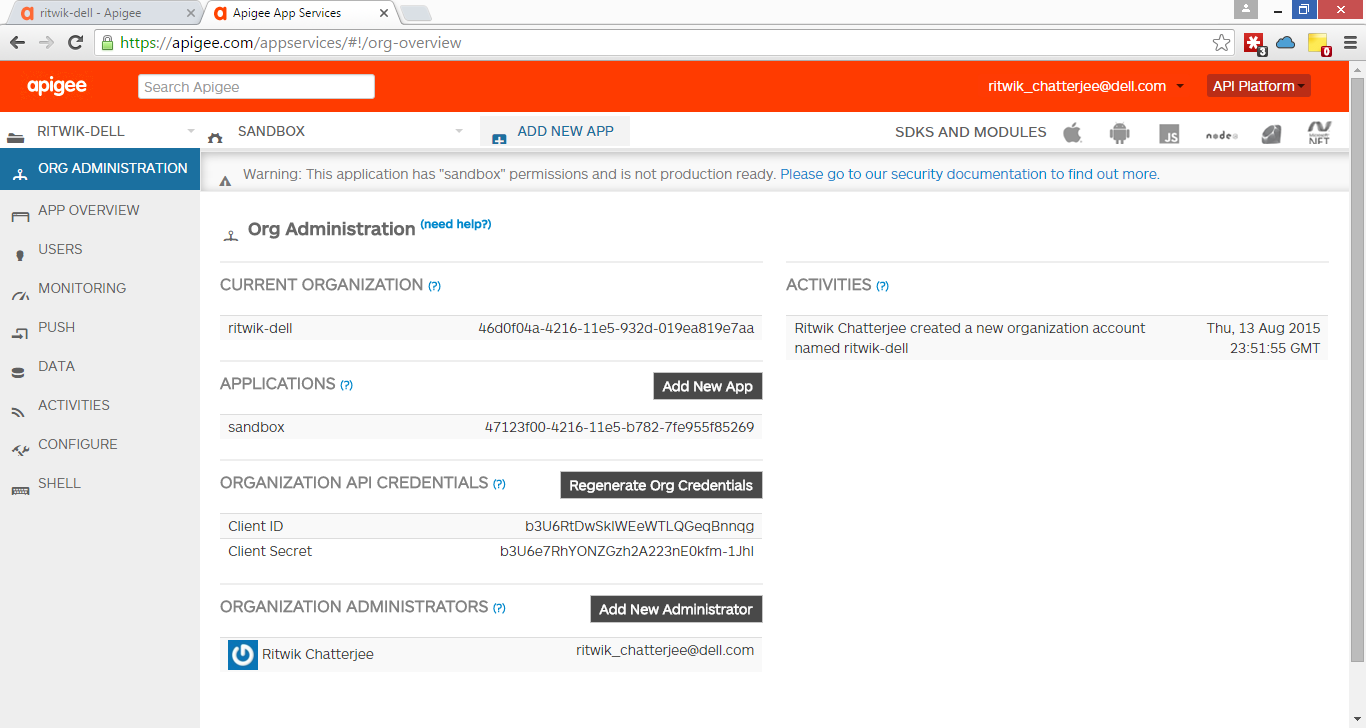
When the server package is downloaded it:

* Creates controllers for each path e.g. /controllers/Products.js
* Services for each controller (/controllers/ProductsService.js)
* Products service implement an example automatically so that the API can return something

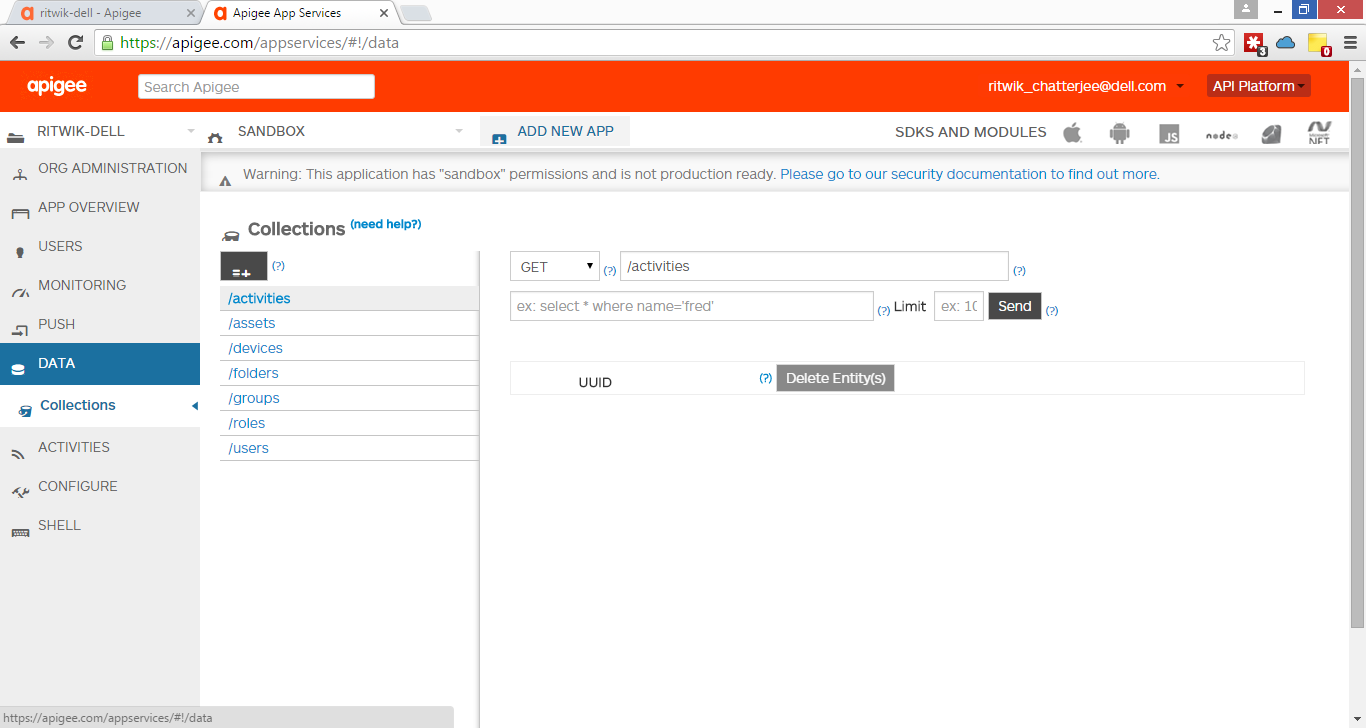
Apigee BaaS information

* Note
  + Node library for BaaS is called UserGrid
  + It is Cassandra with APIs as a front end

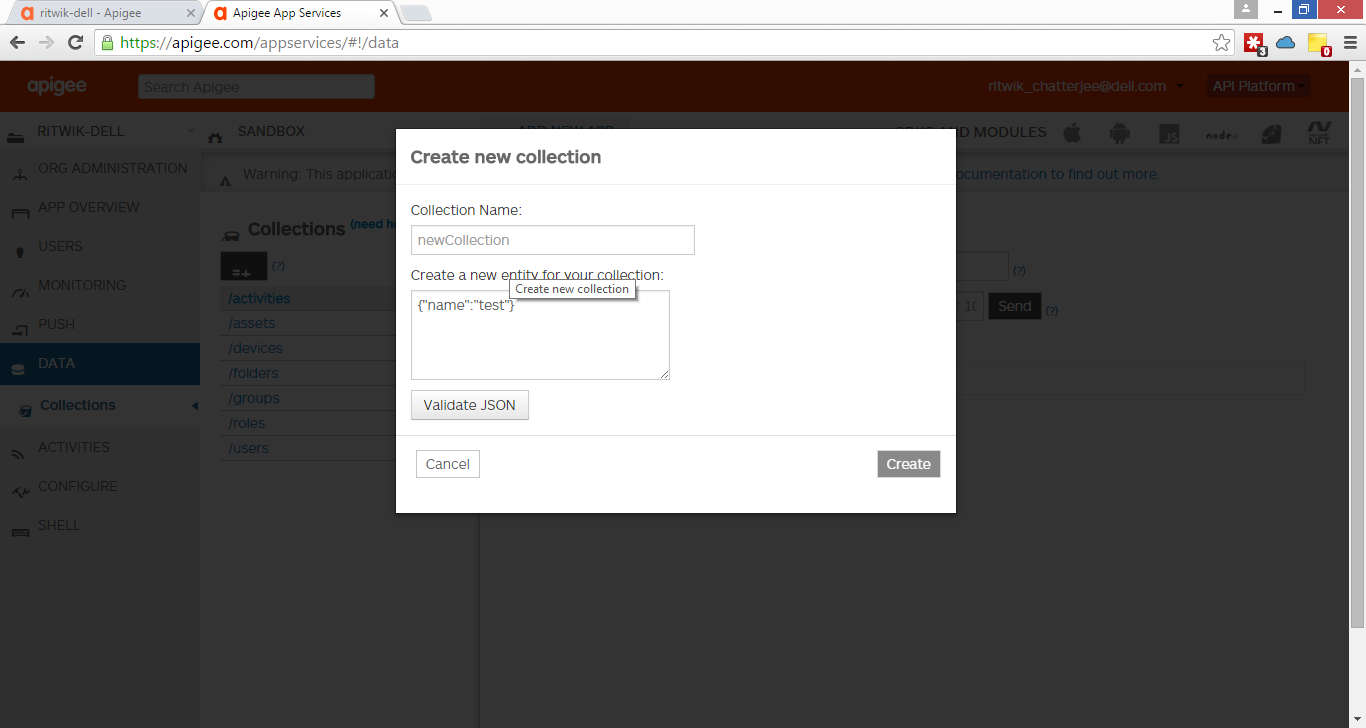
BaaS landing page – mostly meant for maintenance/management tasks

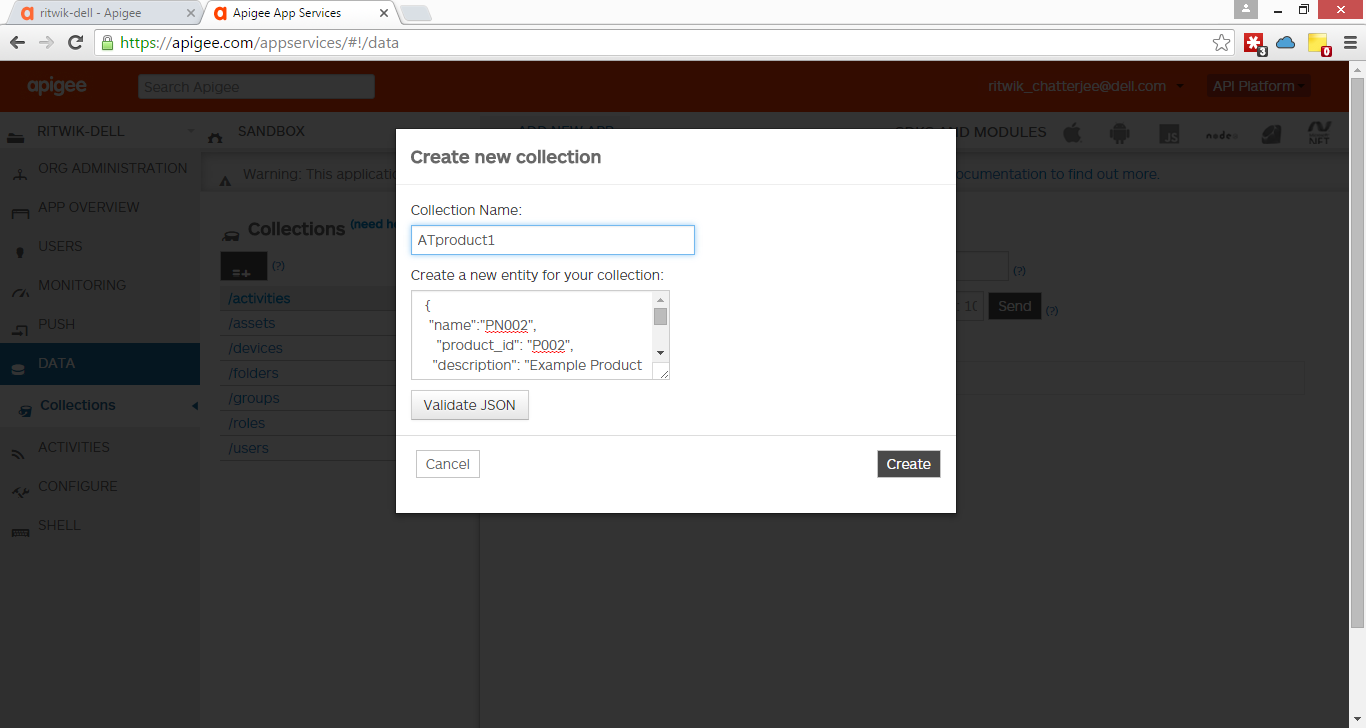


Creating Collection



Adding new collection





* BaaS limitations\*\*\*\*\*\*\*\*
  + Note “name” is a reserved field in BaaS. Meant to be used as a unique string that BaaS identifies a particular record (primary key)
  + There is no way to delete a collection from BaaS. You can delete the data in the collection, but never delete the collection

Integrating with UserGrid/BaaS

* Requires user id and password to talk to BaaS
* Create a BaaS Adapter that will establish connection
  + Since nodejs is all async calls, requires call backs and leads to “Christmas-tree” issue
  + You can use Async library to reduce Christmas tree
    - Still does not remove Christmas trees
  + Use Promises library
    - Its taking the callback loop, makes the functions promise to return a value and once it returns, do the next steps
    - So actually making a async operation, work synchronously – waiting for async to complete
    - Promises is the best approach in terms of understanding from a sync flow programming background perspective because it is easy to get the call-back loop wrong

Installing usergrid

npm install usergrid --save

“--save" saves the library in package.json file.

Usergrid documentation available on npmjs.com, github and Apigee website

Note: Using npm install usergrid was installing usergrid 2.0, which is not what is desired. Desired is an earlier version usergrid 0.11. In order to install that, we:

* Removed the usergrid folder from node\_modules
* Updated package.json with the required version of usergrid
* Ran npm install, which read the package.json file and installed necessary packages

<https://github.com/usergrid/usergrid/blob/master/sdks/nodejs/readme.md>

Documentation from above shows how to declare a client for your BaaS – need to add this to the ProductsService.js

Then use the client to create a collection – and call back on the

Best Practice Note: This is not a good way to do it as you have credentials in code and since it is auth\_client\_id, you are passing it over the wire all the time. Best way to do this is create a BaaS adapter and use it as a singleton. The BaaS adapter should use a new developer user credentials for logging in, and store the user credentials in vault.

Returning only the data that we need from BaaS

BaaS returns a lot of meta-data along with the data requested. To fetch only the piec that we need:

* Option 1: Store the entire JSON under another level, for e.g.
  + Instead of storing {“ProductId”:”P001”,”Product\_Name”:”Product 1”}, you store as
  + {“data”: {“ProductId”:”P001”,”Product\_Name”:”Product 1”}}
* Option 2: Using Object Mapper and extracting individual fields.

IDE for Node.js – IntelliJ WebStorm

Using Object Mapper in Node.js (similar to dosser.js)

* Create a model file (e.g. Products.js) in a models folder with the mapping
* Import these maps and the object mapper in your code

**Deploying Node.js application on Apigee Edge**

Tool in npm download which allows you to push stuff to apigee – found in npmjs.com

Install by

npm install –g apigeetool

Might require admin access

Command to deploy node application

apigeetool deploynodeapp

Asks for Username, password, environments and organization. Note: for username use your registered email address. Else you will get HTTP 401, unauthorized.

The apigeetool deploys the application as an empty proxy linking to the application. By default it sets is up a basepath of “/”.

Help with apigeetool: apigeetool deploynodeapp –h

**Day 4**

**Apigee BaaS Overview**

* Overview
  + Leverages Apache Usergrid with Cassandra database
  + User data – manage users and preferences
  + Push notification
  + Connections and social
  + Does not need a license, but a separate module
* Data Storage
  + Cassandra data store
    - NoSQL handles consistency as “eventual consistency” – I am only going to write data to one place, and DB system will take care that the data is replicated to all places that is required.
    - Fast writes to the data store
    - Schema-less data model (flexible)
    - Linear scalability
* Security & Authentication
  + Provides Client Credentials and Resource Owner Password Credentials grant type for OAuth out of the box
* Entities
* Collections
* Data Modelling
  + Cassandra supports graph maps/data models
* Queries
  + Allows a lot of queries – direct firing of queries (with certain restrictions)
* Matrix Queries
  + Allows you to use embedded query strings in the URL – multiple filters at multiple levels.
* Geo-location
  + You have to store entities with their locations (lat, long)
  + BaaS has the application logic to return entities which satisfy a given selection criteria (e.g. 5000 meters from device geo-location)
  + It does NOT give you the actual ground distance – just the radial distance
* SDK support
  + Freely available for multiple platforms
* Apigee BaaS
  + [https://api.usergrid.com/<org-name>/<app-name>/<collection](https://api.usergrid.com/%3corg-name%3e/%3capp-name%3e/%3ccollection)>
  + App-name = Sandbox

Adding Ratings from BaaS

* Create ratings collection in BaaS
* Add data to it
* Add it as a new Target end-point to your proxy
  + Add a new target end-point
    - If you call it baasTarget, it automatically adds the url https://api.usergrid.com
    - Update the url to api.usergrid.com/<org-name>/<app-name>/
  + Add a route rule to point to the new target end-point when the condition is met

Sandeep Marusupaly

BaaS Use Cases

* Storing Reviews
  + Reviews stored internally
  + Product has review mapping
  + User wrote review mapping
    - BaaS automatically attaches connections/links to the master entities as you keep on adding mapping/transactional entities
  + In order to retrieve data with connections use the key work “connecting”

Mashup Usecase

Advanced Mediation

* XML/JSON Differences
  + XML always has a root element, JSON doesn’t Or has a Null element.
* XML to/fro JSON conversion
  + XML to JSON a little difficult
    - Data types has to be guessed
    - Arrays are difficult and often results in incorrect conversion
  + JSON to XML
    - Little better
  + The JSONToXML or XMLToJSON policies in Apigee has some additional properties that can be tweaked to add some rules
* JSONToXML Policy
  + Additional Policy options
* XMLToJSON
  + RecognizeNumber, RecognizeBoolean, RecognizeNull
  + NullValue

Analytics

* Mechanism
  + Metadata captured in raw from each Message processors in batches
  + Aggregated data by the Analytics Engine used for reporting
  + 17-18 parameters captured by default for analytics
  + Custom elements can be captured and pushed to analytics engine
* Logging to Analytics
  + Dimension and Measure
* Operational Analytics

Edge Micro

* Add-on to Apigee Edge
* Lightweight gateway component, hosted on a separate host to capture analytics/PEP without bringing in the heavy weight gateway

Developer Services/Portal

Monetization

* Publish 🡪 Packages
  + Another level of abstractions on top of products, that includes rate plan
  + Define the rate plans and tie it to products and developers (if required)
* Once packages are defined, developers will need to subscribe to packages
* There is a monetization tab which has links to
  + “Billing Documents”,
  + “Monetization Reports”,
  + “Adjustments”,
  + “Credits”,
  + “Refunds”,
  + “Billing Document Configuration”
* For each product that you define, you need to define “Transaction Recording Policy” – which is basically defining what in your business equates a transaction to be considered as a monetized transaction
  + Transaction Attributes can be based on one or more of the following:
    - Request payload value
    - Response payload value
    - Response status
* Monetization limit check is a policy which checks for monetization
* Looks pretty flexible in terms of what it can offer. Not sure about robustness or accuracy. They claim it is certified.

Smart Docs

* Separated from Gateway and development cycle
* Differs from Swagger UI as in Smart Docs renders Swagger in a different way creating new pages for each new resources
* There is a Swagger2Api command line tool for generating Apigee API Proxy from Swagger specification <https://community.apigee.com/content/kbentry/8796/swagger2api-a-nodejs-command-line-tool-to-generate.html>
* Import of Swagger documentation happens from Developer portal as an admin
  + Rendering can be modified by editing a css
  + Drupal
* Management APIs are there to import Swagger specifications into Smart docs

Day 5

Continuous Integration

Continuous Delivery

Puppet Module Unit tests 🡪 Platform tests 🡪 Deliver to staging 🡪 Application Acceptance Tests -- (manual) 🡪 Deploy to Production 🡪 Post Deploy tests

Mostly use some kind of automation/orchestration to move between the steps. Tools used:

* Puppet
* Jenkins for execution and validation of tests

There is a pretty good ebook by Puppet.

Apigee Support

* Apigee provide all management capability through management APIs, which can be written into CD/CI scripting
* Best practice for moving code through environments
  + Move code deployed in lower environment to higher environment (not deploy code directly into higher environment)
  + Use logical bindings for environment variables in the code, so that it works seamlessly.
  + Only when above is not available/could not be achieved (10-15% of cases as per Apigee), use apigee maven plugin which allows you to declare pieces of code that should change

Troubleshooting

* Splunk/Syslog Integration
  + Use Message Logger to push information to splunk/syslog
  + Different levels of logging possible (INFO, DEBUG, ERROR, etc.)
* You can run a curl command to set the system level variable LogLevel to a certain level (management API)
* LogLevel
  + It is a system level variable
  + By default, set to ERROR in each environment
  + Can be changed using a management API
* Logging Policy
  + Can use flow variables for logging or for a little complex logging, use a JS policy to create message for logging
  + You can optionally add the condition in the step for logging which will run only if LogLevel has a certain value, e.g. DEBUG. If you do not set this condition, it will always run.
  + As per Sandeep, you can create a java script policy to write different contents to log when the log level is at different levels – so if ERROR, write short, if DEBUG, write details – and then let this whole thing run under all conditions.

Caching

* Apigee can offer, **Apigee DN** (distributed network) – geographically dispersed data center
  + Scalability can be achieved only if the data is cacheable
* Where
  + L1 – in-memory on MP
  + L2 – on disk – Apigee Cassandra for distributed caching
* Policies
  + Response Cache Policies
    - Has to be attached on both request/response for lookup and population (same policy)
    - Entire HTTP response (including headers and metadata)
    - Can set a different time-to-live for each entry
    - Option for honoring HTTP cache headers for dynamic TTL
    - Option to support caching of multiple formats based on request Accept header
  + Populate Cache/Lookup Cache
    - Full control over caching, store any objects, partial caching
* Response Cache
  + Cache key
    - Can use multiple key fragments
    - Scoping affects the cache key (flow variables)
  + Expiration
    - Can specify a time of day or date the cache entry expires, or direct TTL
    - Leverage <UseResponseCacheHeaders> configuration so the policy dynamically sets TTL based on back end response HTT cache headers
    - Can invalidate the cache externally (specific cache or all cache)
  + Optional condition for skipping cache lookup or population \*\*\*\*
    - Skipcachelookup or Skipcachepopulation
  + Have to attach the same policy both on request and response path for lookup and population
    - If cache hit, response is retrieved from cache and processing bypasses backend and other policies until ResponeCache policy in Response segment \*\*\*\*
  + CacheResource
    - By default there is a default cacheResource
    - But also can create your own cache Resource, which allows you to set parameters like limiting the size of the cache
    - Separate for each environment
  + Scope
    - To whom are the cache is available
    - Values can be
      * Application – proxy level
      * Organization –
* Populate Cache and Lookup Cache policies
* Clearing the Cache
  + UI – to invalidate ALL cache entries
  + Management API – typically to invalidate ALL cache entries
* Invalidate Cache Policy
  + Allows you build a fully customized AP resource to invalidate specific entries
  + Used in implementations to invalidate specific cache entries
* Analytics
  + Analytics 🡪 Cache Performance

Tips

* Extract Variables
  + URI path name should begin with path-suffix, should not include base path
  + If you populate the <variable prefix>, it will not populate in the trace\*

Fault Handling

* There is a request flow and a response flow. There is also a fault flow – which starts from the fault and goes back to the client.
  + On the trace you see this superimposed on the response flow.
  + Default fault flows – for all
  + Other rules
* In HTTP world anything that is not 2XX, 3XX is fault
* Fault can happen either at external call or any internal policies
* When a fault happens, all subsequent policies get executed and the control goes to Fault Flow
* Aspects of Fault Handling
  + Policy Faults: Some policies allow definition of fault message to be raised for a fault
    - Fault Message can be customized for some policies
    - Fault can be suppressed for some policies (e.g. related variables continueOnError, ignoreUnresolvedVariables)
      * You can still check for the fact that fault happened by looking at policytype.policyname.failed? variable which will be set to true in such a case
* In Fault flow
  + The Fault pipeline gets executed
  + <FaultRules> is used to define the fault flow (bottom up approach)
  + Or <DefaultFaultRules> to define the flow for ALL faults
* Defining the Fault-flow
  + Steps and conditions for each step
* Raising a custom fault
  + Policy called RaiseFault

Apigee Sense

* Looks for traffic patterns to detect botch traffic

**Questions**

1. How API Key is validated? Where are the API Keys stored? How is it secured? How expensive is the operation? Where should it be in the flow?
   1. Stored in Cassandra – passed PCI compliance tests
   2. But since API key is not considered an absolute secret not much focus on securing the information
2. Which component works as the Authorization server for the OAuth flow?
   1. Message Processor creates the token and stores it in Cassandra
   2. However can integrate with external authorization server – introduces
3. Where are the KVMs stored? How are KVMs secured?
4. How can I bake in MFA?
5. CI integration
6. Can Apigee be integrated with enterprise LDAP/User data-store?
7. Support for encryption?

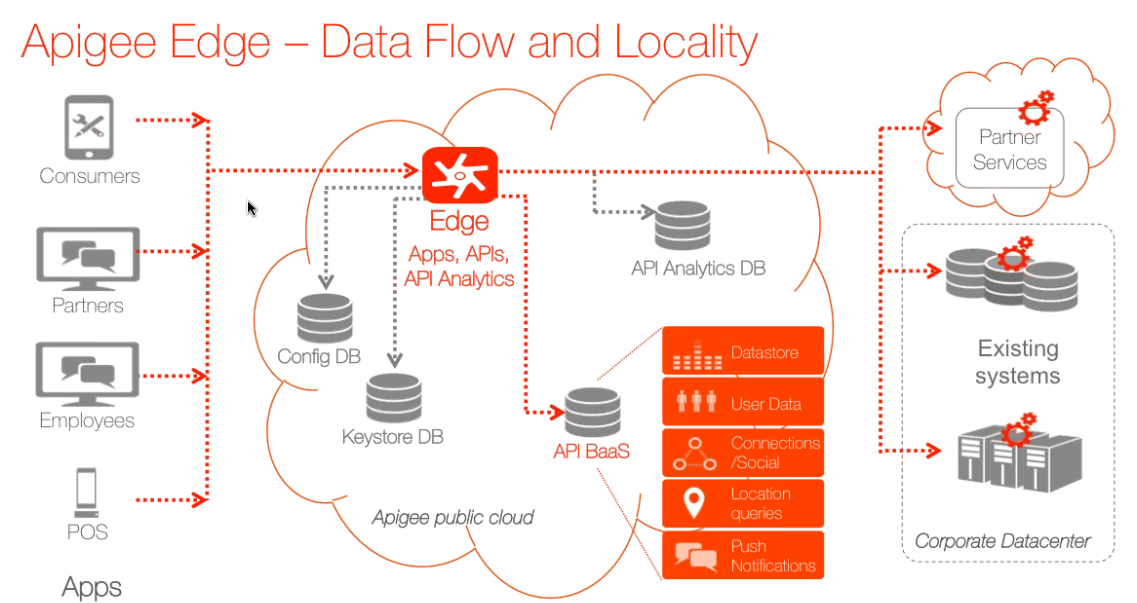
Work Experience

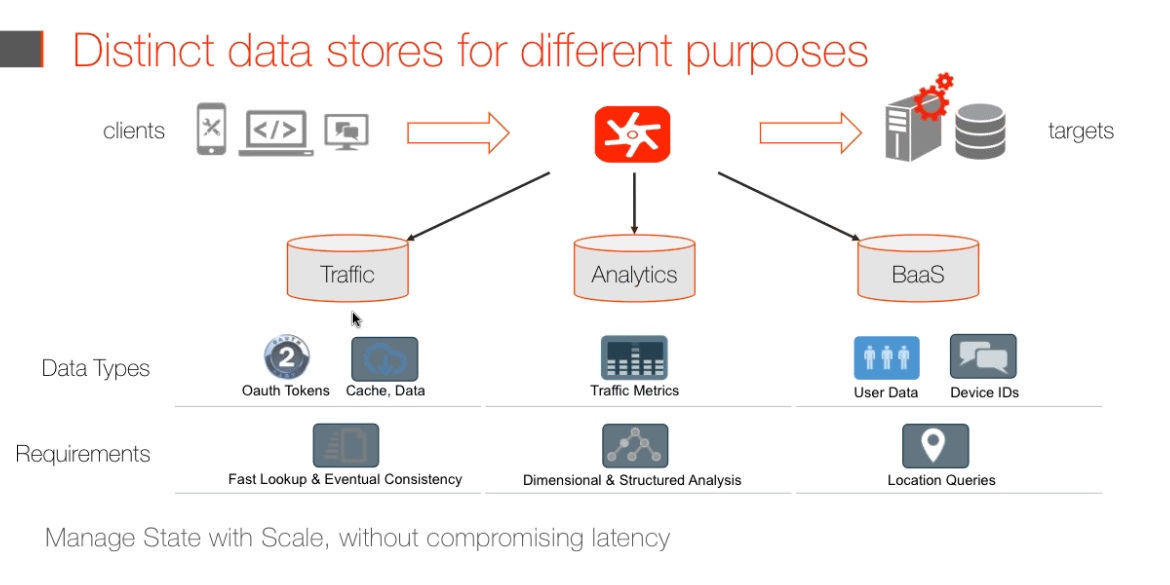
**Swagger2api**

* NPM Tool used to convert Swagger definitions into API proxy bundles for Apigee
  + <https://www.npmjs.com/package/swagger2api>
* Current tool version 2.14.12
* Training:
  + <https://community.apigee.com/articles/8796/swagger2api-a-nodejs-command-line-tool-to-generate.html>
  + <https://community.apigee.com/articles/9478/swagger2api-020-version-generating-apigee-policies.html>
  + <https://community.apigee.com/articles/9741/swagger2api-021-version-securing-apis-using-swagge.html>
* Tool Limitations:
  + Limitations with $ref references. Treats it in a different way that Swagger UI

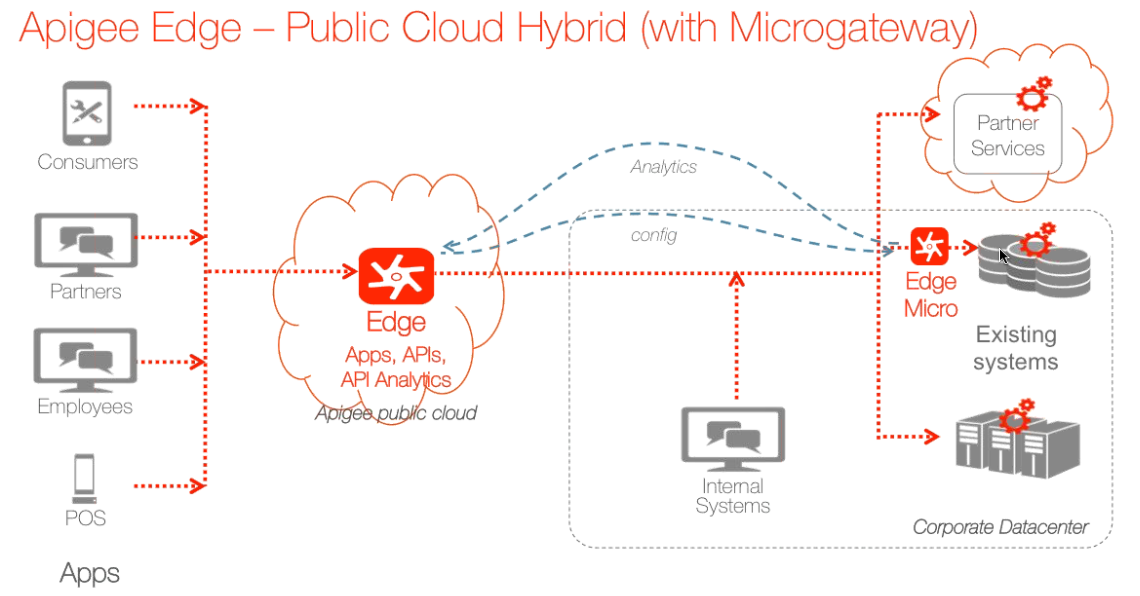
**Apigee Edge – Architecture Overview (Apigee SE Bootcamp)**

November 21, 2016



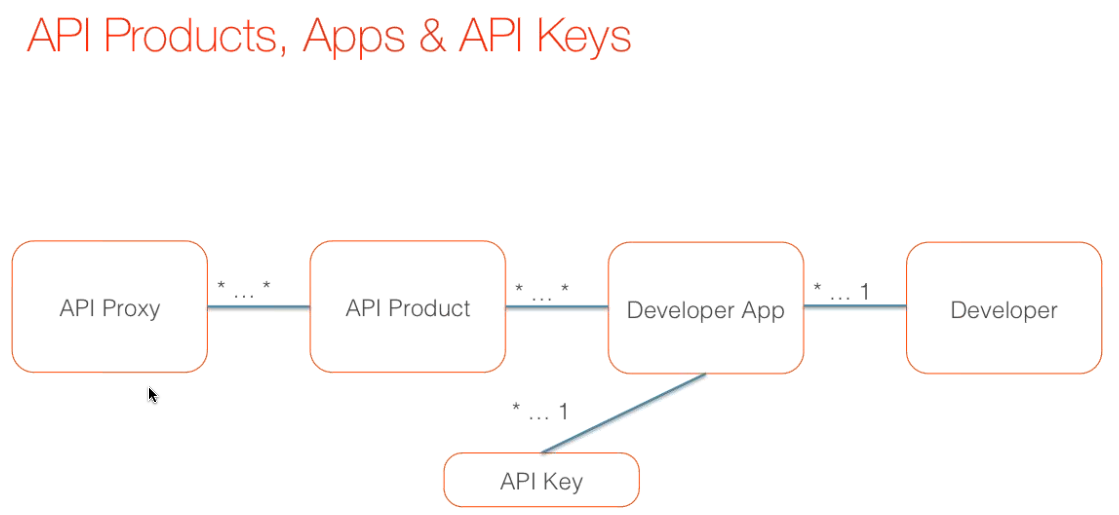


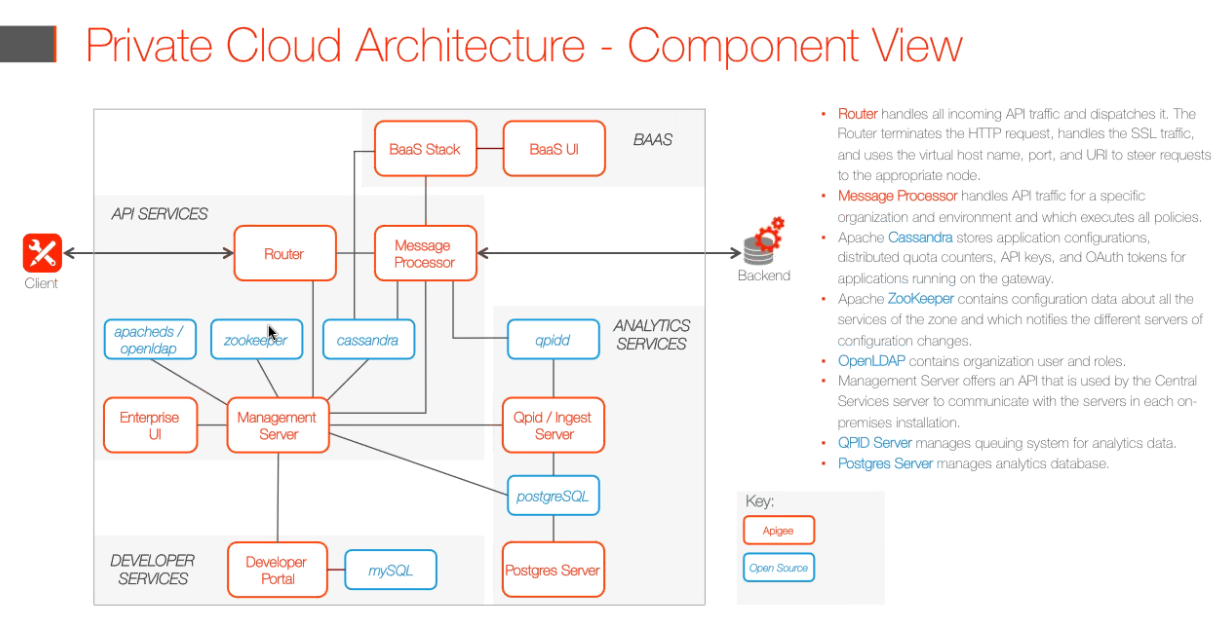
* Traffic – Cassandra
* Analytics – PostGres
* BaaS –



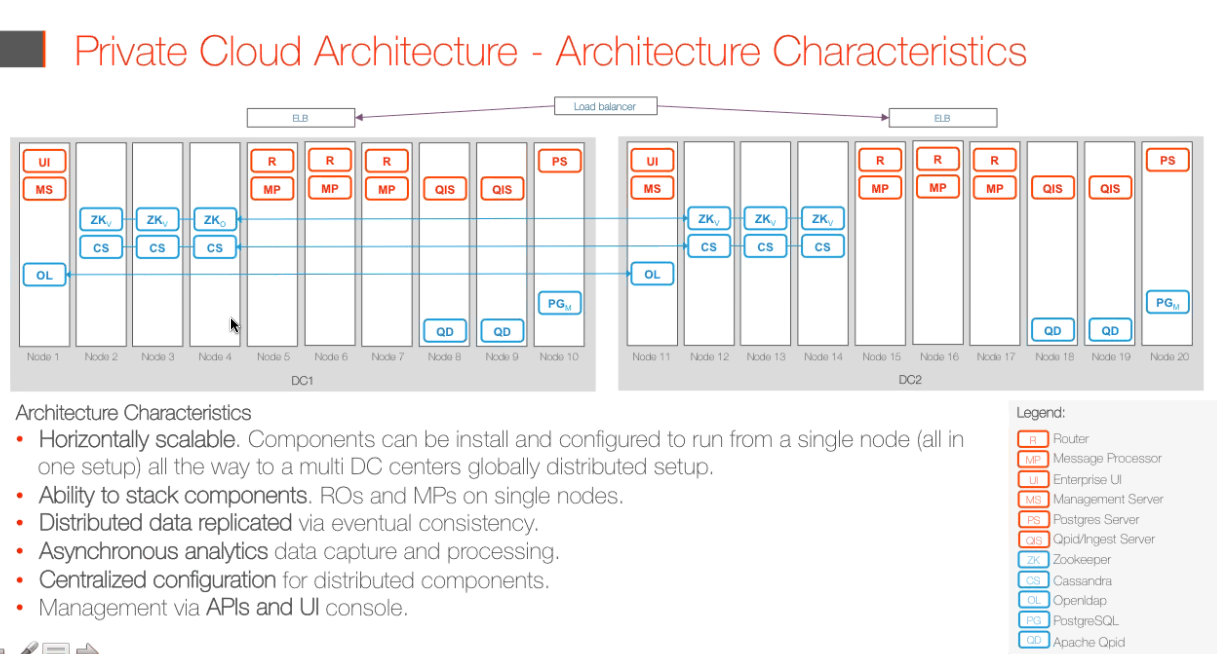
Most common use-case:

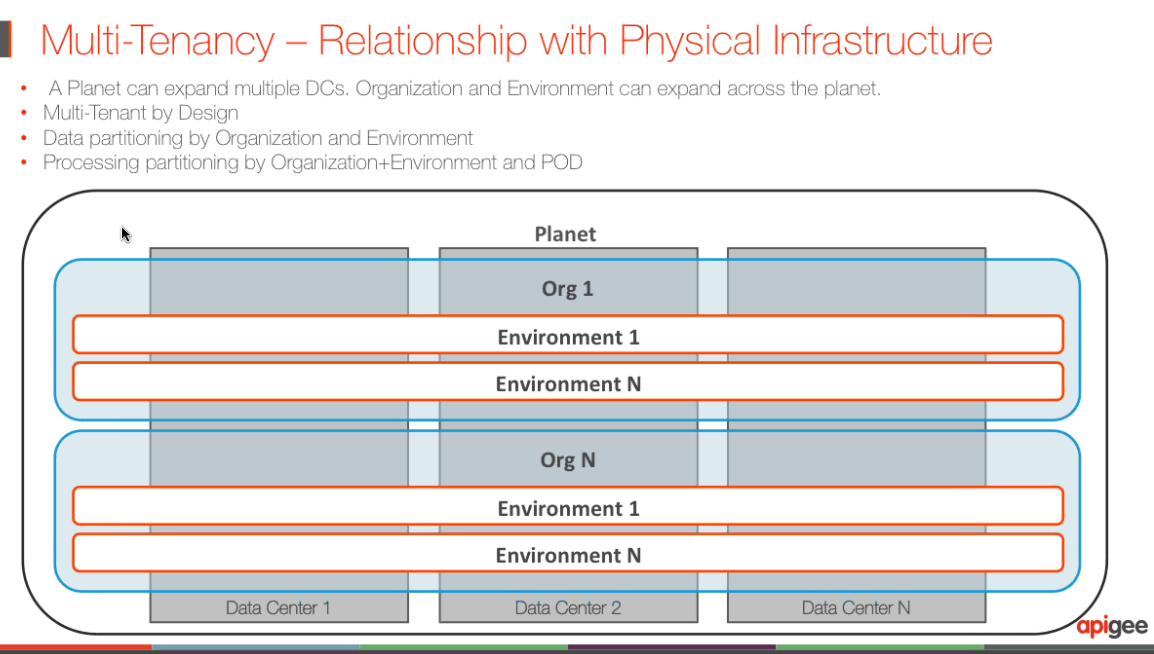
* Where microservices are being used where microservice intercommunication
* Edge-micro does minimal API management (light-weight) –
  + will pull config from central product
  + send analytics back
  + can be deployed globally

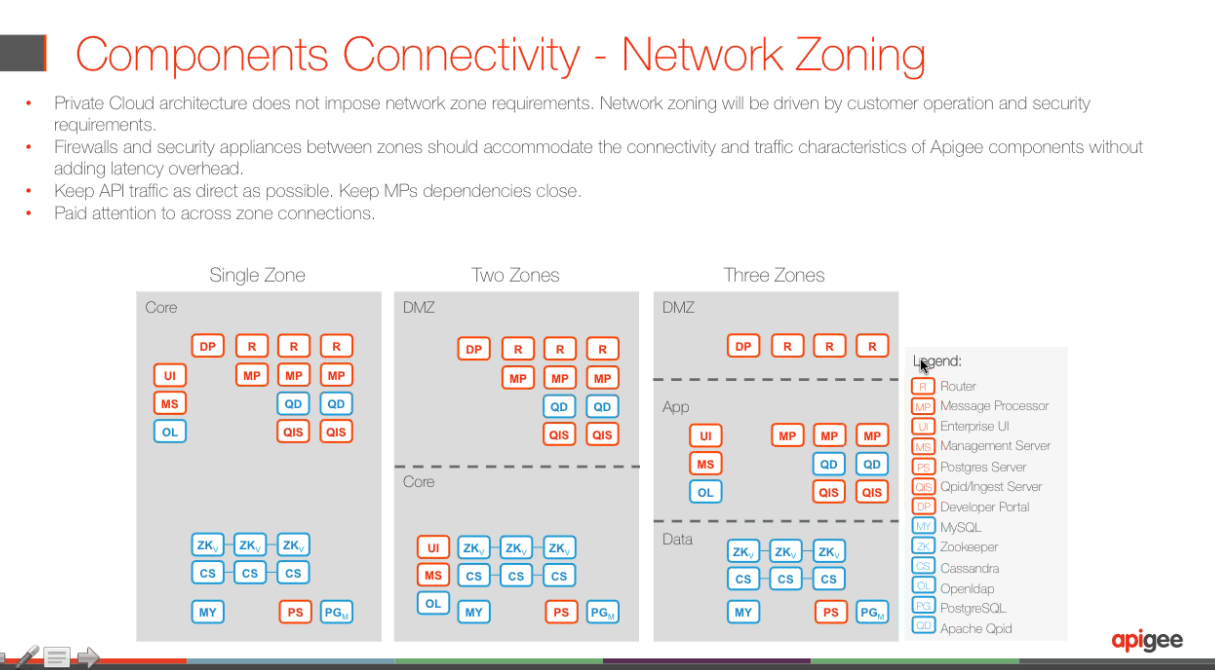




* Typically you will have Router and Message Processor on the same host. However, when you have a lot of heavy lifting happening in proxy, split them across
* Developer portal is set separately because it can be setup separately in a different subnet
* Gateway = router + Message processor
* For scalability you need to have multiple gateways, while others can remain the same
* Data Center redundancy
  + Gateways can be working at different DCs
  + No master slave – all master-master







**Competitive Differentiation**

Joshua Norrid [Jnorrid@apigee.com](mailto:Jnorrid@apigee.com)

* Apigee Developer portal – is fully customizable (open) – Portal is built on Apigee Edge Management API. Don’t need to engage Apigee Professional services
* Apigee provides complete language bind for multiple language extensions – argued as the only one in the market doing this
* Zero downtime for changes in a global deployment (use of Cassandra)???
* Analytics
  + Async
  + Dynamically created
* Analytics capabilities is a strong differentiation

