C855 Introduction to REST APIs and Blue Planet Orchestrator APIs

Casey Boyles
Technical Learning Developer
Ciena Blue Planet

Introduction to REST APIs

Objectives:

- **Understand Services**
- **Define REST**
- **Define API**
- **Identify the characteristics of REST APIs**
- **Identify the key benefits of REST APIs**
- **View REST API Demonstrations**

BPI API Overview:

- **BPO Core API's**
  - **Authentication**
  - **Market**
  - **Products**
  - **Resources**
- **Core Tools**
  - **Swagger**
  - **cURL**

About Services, REST, and API’s

Internet Based Services

- A web service is any piece of software that is made available over the internet
- Use a standardized messaging system such as JSON or XML
- Are self-contained, modular, and distributed
- Are based upon open standard protocols such as TCP/IP or HTTP
- Built using any language such as Python, Java, JavaScript, or FireChicken
- May be used to invoke processes, create products, control data flow or supply chains

Data Exchange and Open Interaction

- Web services use information exchange systems, on the back of Internet Protocols, to facilitate direct application-to-application interaction
- These applications may include documents, messages, objects, processes, programs, or services
- Web services are agnostic about the languages the applications are written in (Python, Java, C, FireBiscuit)
- Web services are agnostic about the platforms they run on (Linux, Unix, Windows, AS400, Blue Planet)
- This allows for data exchange, using open standards amongst heterogenous systems

Services and REST

- REST is one of many implementation mediums that may be used to create Internet services
- Another popular medium is Web services (based on standards, WSDL, and SOAP)
- REST based services have distinct characteristics
  - Enforce a number of design and architectural requirements on the service contract and logic
  - Does not restrict how the service is designed nor how the service may be purposed
- Therefore, REST gives developers a bounty of freedom as to how the process that is used to model and design REST services

What Is An API?

- An API (Application Programming Interface) is an interface to an application or a service
- This interface exposes data and functions to facilitate interactions between computer programs
- These interactions are designed to allow systems to exchange data

What is a REST API?

- It is common to find the REST architectural style applied to an API designed for modern web services
- An API that conforms to the REST architectural style is a REST API
  - Therefore, REST API's make a web service RESTful
- RESTful applications expose a consistent interface for developers to develop with

What is REST?

- REST Stands for Representational State Transfer
- REST is an architectural style for distributed hypermedia systems, and is a hybrid of many network-based architectural styles
  - Focused on creating Resource based services, opposed to action based systems
- Term Coined in Roy Fielding's dissertation "Architectural Styles and the Design of Network-based Software Architectures" (2000)
  - The dissertation lays out a series of constraints that should be in place when 2 systems communicate
- Essentially, REST is simply a series of rules, in place for a service, so that service consumers understand what the service does and how the service works

Resource Based System

- Resource based systems are focused on providing resources (things) opposed to performing actions
- REST focuses on Nouns in a system opposed to verbs (Data vs Methods)
- Resources are identified by URI's (Uniform Resource Identifiers)
- URI calls are implemented by HTTP methods (verbs) such as GET, POST, PUT, DELETE
- The HTTP methods dictate the types of operations that are performed
- Multiple URI's may point to the resources and expose different representations of the resource
    - Such as a GET request may fetch a resource where as a POST may create a resource or a DELETE may remove a resource

Resource Representations

- Resource representations, represent the state of the resource on the server
- Predicated on the type of HTTP Request, the representation may represent the whole resource, part of a resource, an insertion of a resource component, the update of a resource component, or the deletion of a resource component
- The data exchanged between client and server are typically JSON based or XML based, but may be any agreed upon data exchange format such as CSV, Text Tab Delimited, or Plain Text

The Need For REST

- The REST Style originated as a description about how the web, and web applications should work
- The impetus was to standardize the web based protocols, and to formalize the structure of internet based applications and the web
- The need for common standards are also important so that business applications may be built on said standard and allow for interoperability among them
- Common standards also developers to develop applications using a common approach, and common methodologies

The HTTP Object Model

- The HTTP object model was originally designed by Roy Fielding to describe how applications would work based on the HTTP specifications
  - This allowed for the ability to map application behavior to the HTTP standard
  - This also allowed for the ability to describe and forecast how changes in the HTTP standard would affect Web based applications
- Roy studied up on software architecture, reproached the HTTP Object Model, and the result of the efforts is the REST Architectural style
- Essentially, REST describes how HTTP works from a data/behavioral standpoint for software architecture

REST Features (Constraints)

REST Constraints (Features)

- Roy Fielding's dissertation sets out several constraints that must be in place in a REST based system
  - Client-Server
  - Stateless
  - Cache
  - Uniform Interface
  - Layered System
  - Code-On-Demand (Optional)

The Uniform Interface Constraint

- The uniform interface constraint defines the interface between the client and server
- REST uses the HTTP Specification to implement the uniform interface constraint
- URI’s are used to represent resources (resource names)
- The Uniform Interface constraint is broken down into 4 further constraints
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state (HATEOAS)

Identification of Resources

- Each web based element is known as a resource
- Resources are identified by a URI (Uniform Resource Identifier)
- E.g. http://www.ciena.com

Manipulation of Resources through Representations

- Clients may manipulate representations of resources
- A resource may be rendered differently to different clients
  - Web page to a browser
  - XML to an XML parser
  - JSON in response to a service call
- These different rendering styles are representations
- Fundamentally, a representation is a method to interact with a resource
- This allows a resource the ability to be represented differently, without changing its identifier

Self-descriptive Messages

- The desired state of a resource may be represented within a client's request message
  - E.g. a client may tell the server to send an HTML doc, or a PDF, or a small killer whale
- The resources current state may be represented in the response message from the server to the client
  - E.g. The server submits the response mime type in the response header, and the representation in the body of the packet

Hypermedia as the engine of application state (HATEOAS)

- This sub constraint is a component of REST that separates itself from other network architectures
- This constraint specifies that the client interacts with the application, entirely through hypermedia that is provided by the server
- These are usually presented as a series of hyperlinks
- The hyperlinks are used to allow clients to navigate through related sets of resources
- The presence or absence of a link may also be useful in describing a resources state

Uniform Interface Client Calls

- Clients typically make HTTP(S) URI based calls to the server
- It is important to note, REST is not constrained to using HTTP. However, HTTP is a ubiquitous and convenient protocol
  - HTTP verbs are used to represent the actions developers take on the resources (GET, POST, PUT, DELETE)
  - HTTP Verbs also simplify calls to a RESTful as they are commonly used, and easy to understand request methods
- On the client side, this has a net effect of decoupling the clients from the servers. Clients do not need to be gnostic of the server as they are making a simple TCP/IP Application layer request
- This also allows the servers to respond to any client call that can make an HTTP call for a URI

Example: Uniform Interface Client Call

- An example request may look something like this: http://ciena.com/contacts/listing?uid=8675309
- With the following request header (partial)

```
GET /contacts/listing HTTP/1.1
Host: ciena.com
Querystring: uid=8675309
```

Uniform Interface Server Responses

- Because REST typically relies on HTTP calls, any standard HTTP server may be stood up to handle requests
- This HTTP Server may be the primary nexus for interaction with the Application, or it may simply be a proxy service to a layered system that is abstracted behind the scenes
- HTTP servers handle the resource request and respond to the client request with standard TCP/IP Application layer packets
- The packets are separated into the Response Header and the Response Body

Uniform Interface Server Response Headers

```
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 5 June 2017 04:36:25 GMT
Server: nginx
Connection: close
Pragma: public
Expires: Sat, 5 June 2017  05:36:25 GMT
Cache-Control: max-age=3600, public
Content-Type: text/JSON; charset=UTF-8
Last-Modified: Sat, 5 June 2017 03:50:37 GMT
```

Example: Uniform Interface Server Response – Response Body Data

```
{
        "data": [{
                "type": "contact",
                "id": "8675309",
                "attributes": {
                        "fname": "Jenny",
                        "lname": "",
                        "created": "2015-05-22T14:56:29.000Z",
                        "updated": "2015-05-22T14:56:28.000Z"
                },
```

Example: Uniform Interface Server Response – Response Body Data (Con't)

```
"relationships": {
                    "contact": {
                          "data": {
                                "id": "8675310",
                                "fname": "Tommy",
                                "lname": "Tutone"
                          }
                    }
              }
       }]
}
```

The Stateless Constraint

- Statelessness is a key component to Representational State Transfer
- The host of the RESTful service should maintain zero client state
- All, state data that is necessary to handle the request is contained within the request itself
  - As part of the QueryString, URI, Body, or Headers
- The URI is used to identify the resource and the body that contains the state or state change of the resource
- Once the server finishes processing the resource, the appropriate state of the resource, or pieces of the state of the resource, are communicated back to the client via HTTP response headers, status, and an HTTP response body
- This concept is like an HTTP session

Processing The Stateless Constraint

- The URI is used to identify the resource and the body that contains the state or state change of the resource
- Once the server finishes processing the resource, the appropriate state of the resource, or pieces of the state of the resource, are communicated back to the client via HTTP response headers, status, and an HTTP response body
- This concept is like an HTTP session

Stateless summary

- Servers contain no state information about the client (Clients Responsibility)
- Requests contain enough context to process each message
- Messages are self-descriptive
- The client must include all information for the server to fulfill the request, resending state as necessary if that state must span multiple requests
- Statelessness enables greater scalability as the server does not need to maintain, update, or communicate the session state

The Client-Server Constraint

- RESTful interactions are client-server interactions
- A uniform interface is paramount in REST to separate clients from servers
- The client-server interaction assumes that the systems are disconnected
- Clients won’t always have direct connections to assets such as databases, or processes
- Maintains defined separations between what is a user interface and a service provider
- The uniform interface is the connection between the client and server (API)

The Cacheable Constraint

- Server responses should be cacheable
- Forms of caching implicit, explicit, or negotiated
- Implicit caching is when the caching scheme is not set and the client caches content
- Explicit caching is when the server specifies caching of the content (Cache-control, Max-age, etc.)
- Negotiated caching client and server negotiate how long an item will be cached

The Layered System Constraint

- In a RESTful system, many layers of client-server interaction may be specified to provide resources
- Clients cannot assume they are connected to the end service provider or an intermediary server
- This allows for N-Tier solutions to be built out in a scale-out or scale-up format
- Also, allows for a separation of concerns
- This scalability allows for extra layers of security checks, load balancing, caching, and a strong separation of duties amongst systems

The Code On Demand Constraint (optional)

- While this is an optional constraint, it is a foundational constraint for many web based applications
- This constraint allows the server to temporarily extend the client
- The server may send any model, controller, view, presentation style logic to the client that the client can process
- Notable example code would be JavaScript (ECMAScript), or Java Applets
- This allows the client to, temporarily process routines that would normally be processed on the server
- Typically this logic is used to operate against a state of the resource representation

REST Constraints Summary

- Client-Server
- Stateless
- Cache
- Uniform Interface
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state (HATEOAS)
- Layered System
- Code-On-Demand (Optional)

RESTful Architecture Benefits

RESTful Architecture Benefits

- Compliance with REST constraints offers the following benefits
  - Scalability
  - Simplicity
  - Modifiability
  - Visibility
  - Portability
  - Reliability

Key benefits of REST APIs

- Client / Server Separation
- Scale out / Scale up live
- Platform and language agnostic

Client / Server Separation

- The REST protocol allows complete separation between the Requester interface, service platform, and data storage
- This allows for a myriad of Requester interfaces to interact with the RESTful application
- RESTful applications may be stored on a single, homogenized server solution, or heterogenous distributed server solutions
- Data for RESTful applications may be stored in single data storage solutions such as a relational database, or across distributed applications such as content management systems, distributed storage area networks, big data solutions such as Hadoop, or end user solutions such as Excel

Reliability, Scalability, Visibility, Simplicity

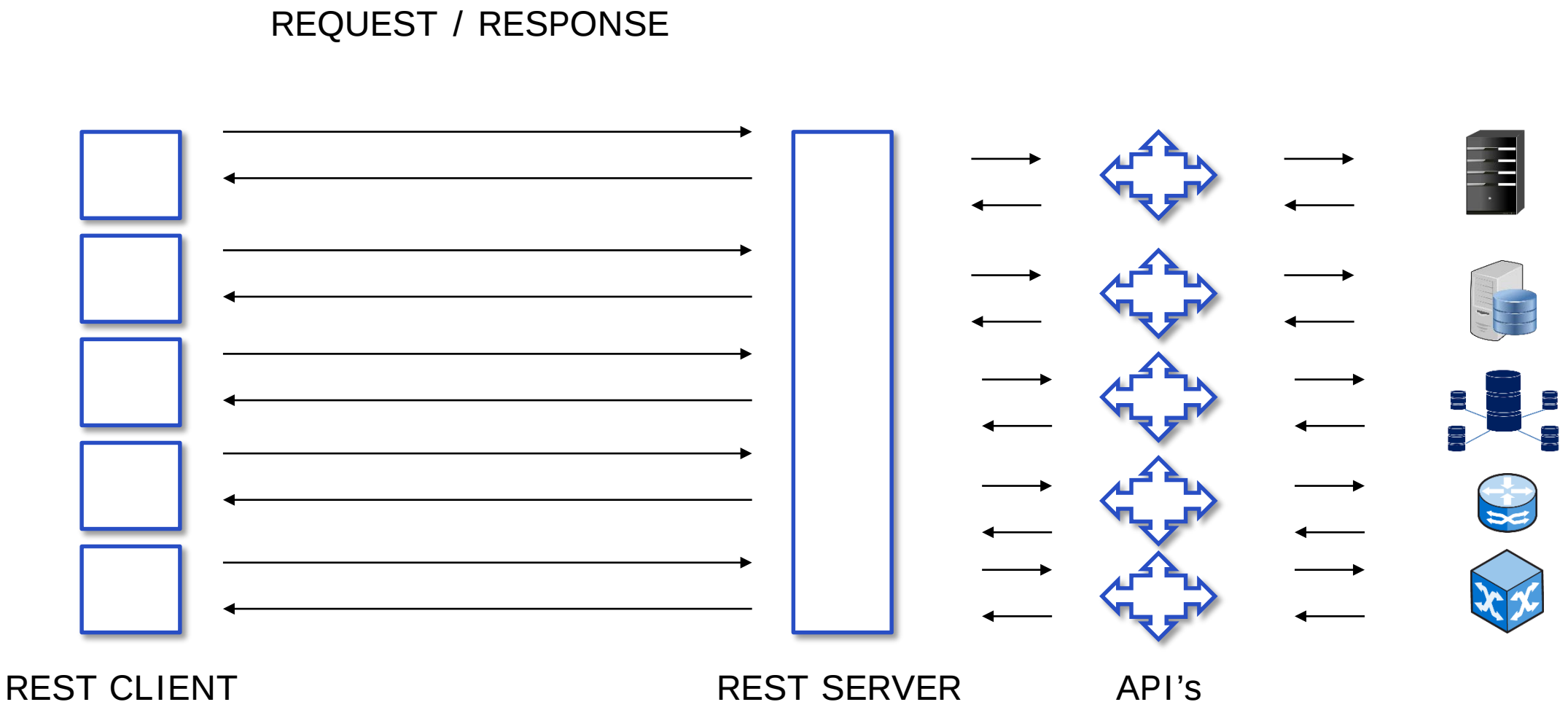
- Separation between client and server allows dev teams the flexibility to scale the solution when necessary
- Data migrations may occur at any time so long as the calls for the data do not change, and the structure of the return data doesn’t change
- Heterogenous clients may be supported since the server is not gnostic of the client.
- Developers may design solutions then to support mobile requests, browser requests, service requests, command line interface requests, api to api requests.

Modifiability, Portability

- REST API’s are platform and language agnostic
- Rest API’s adapt to the type of platform being used
  - Offers freedom to alter or test within the dev environment
- REST API’s may support calls from any type of server or platform (Java, Python, nginx)
  - Makes working in a heterogenous environment, trivial
- The only real constraint, is the client / server should use an open and agreed upon data exchange format such as JSON, or XML

Demonstrations Of REST API’s

REST Transaction Diagram



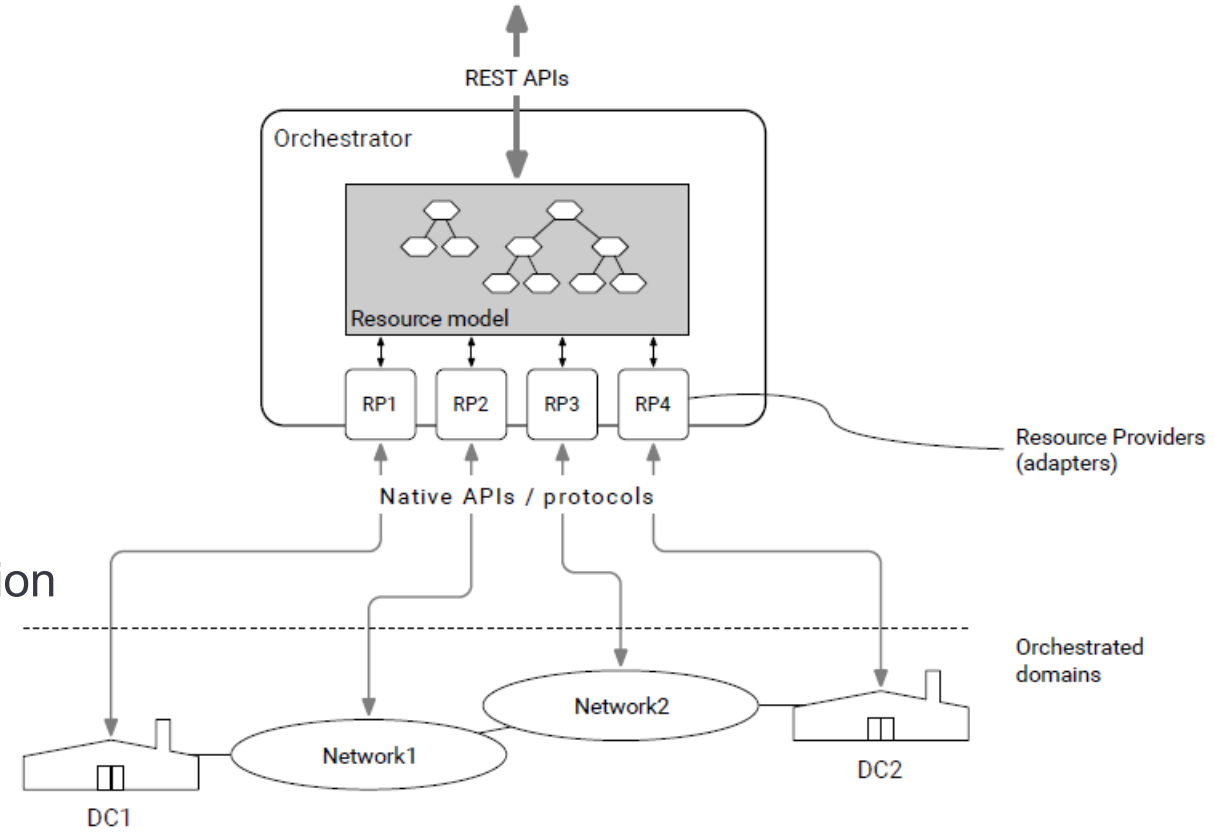
BPO API’s Overview

Blue Planet API Overview

- **Blue Planet Orchestrate is a multi-function orchestration platform**
- **Coordinates and automates digital resources**
  - Creation
  - Management
  - Synchronization
- **Digital resources include**
  - Networking (carrier networks, datacenter networks)
  - Storage
  - Computing
  - Applications
  - Services

Blue Planet Resource Model

- **Blue Planet supports a generic resource model**
  - This allows for the unfired orchestration of
    - Physical
    - Virtual
    - Logical resources
- Planet Orchestrate supports the following:
  - Multi-domain service orchestration
  - NFV (network function virtualization) orchestration
  - Network orchestration
  - Cloud orchestration



Blue Planet Orchestrator Components

- **The Blue Planet orchestrator is comprised of many different components**
- **Each set of components exposes its own set of API's (Application Programming Interface)**
  - Each API is accessible through REST calls
- **API's allow for the ability to execute actions with the orchestrator through simple REST calls**
- **The API's also allow for external applications to perform actions with BPO through simple REST calls**

BPO API Groups

- **Blue Planet API's are comprised of two groups**
  - BPO Core API's
  - BP Solution API's
- **Core API's are API's that operate on**
  - Resources
  - Resource Types
  - Domains
  - Domain Types
  - Resource Providers
  - Etc…
- **Solution API's are provided with solutions and operate against**
  - Alarms
  - UAC
  - Etc…

Connecting to Blue Planet API’s

Connecting to Blue Planet API's

- **Any tool that can make a REST call may be used to connect to Blue Planet API's**
- **Common means to interact with Blue Planet Orchestrator include:**
  - Using an HTTPS URL to perform a REST call using
    - cURL
    - Wget
    - Postman
  - Swagger
- **This course will focus on using cURL and Swagger**

URL Format for REST Calls

- **There is a common URL pattern that may be used for access to BPO API's**
  - This pattern is based on the Blue Planet REST API Guidelines, and common HTTP URL structure

Format:

```
<protocol>://<host>:<port>/bpocore/<api_component>/api/v1/<api><command_parameters><general_parameters>
```

- **Format explanation**
  - `protocol = HTTPS`
  - `host = Blue Planet host IP address or domain name`
  - `port = 443`

Example:

```
https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000
```

More Common Rest URL patterns

- **Requests to Resource Endpoints follow the following conventions**
  - Paths for requests may resemble the following patterns
    - `/bpocore/<api_component>/api/<version>/<api>`
    - `/bpocore/<api_component>/api/<version>/<api>?<command_parameters><general_parameters>`
    - `/bpocore/<api_component>/api/<version>/<api>/<id>/<resource>/`
  - Paths should end with trailing slashes to maintain relative URI's
    - `/bpocore/bpoapi/api/v1/myresource/`
    - `/bpocore/bpoapi/api/v1/myresource/?param1=value1`
    - `/bpocore/bpoapi/api/v1/myresource/1234/?param1=value1`
  - `https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000`
  - `https://10.41.89.52/bpocore/market/api/v1/resource-types`

Example: Using An Authenticated Call to Return A List of

- **In this example we use an authentication header to generate a list of resource types from Blue Plane orchestrator (more on authentication headers later in the course)**
  - This example uses the cURL command, a command line tool

```
curl -X GET --header "Accept: application/json" --header "Authorization: Bearer
8bc1afd3d29d35a54ef8"
"https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000"
```

Common Parameters For REST Calls

- **Query String arguments may be supplied with the REST call to alter the composition of the request**
- **These arguments are supplied as `key=value` pairs that are concatenated together by an ampersand on the right side of the query string character `?`**
- **Common parameters include**
  - `q`
    - Optional query parameter to define a query filter using "property:value" syntax
  - `p`
    - Optional query parameter to define a partial string match filter using "property:value" syntax
  - `offset`
    - Requested offset within the total result set to be the first element in the paged response
  - `limit`
    - The maximum number of elements to return in a single paged request

Example

```
https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000
```

Common HTTP Request Types

Operation	HTTP Method	Result	Description
Query (Read)	GET	200 - Single objector paging of objects 404 – Resource not found	Used to retrieve a representation of a resource
Create	POST or PUT	201 – Newly created object with its attributes (POST) 200 - Object with its attributes (PUT) 400 - Invalid parameters	Used to create a new resource using the specified resource as a factory. The created resource is a top-level resource
Update	PUT or PATCH	200 – Newly updated object with the new attributes 400 - Invalid	
Delete	DELETE	204 - No message 400 – Invalid parameters 404 – Resource not found	Used to remove a top level resource

For Example:
http://ciena.com/contacts/listing?uid=8675309

```
GET /contacts/listing HTTP/1.1
Host: ciena.com
Querystring: uid=8675309
```

Common HTTP Status Codes

Status Code	Descripton
200	Success. Used for GET, PUT, and PATCH
201	Created. Used for successful POST request
204	Success with no content. Used for successful DELETE request or for GET requests where a valid query matches no content
400	Bad request. This is used for validation error and other known verification. The key here is that these are "known" exceptions
401	Unauthorized. Failed Authentication
403	Forbidden. Permissions denied
404	Resource not found. Used when accessing unknown resource
500	Internal server error. This is used when handling unknown exception

Example:

```
HTTP/1.1 200 Connection Established
StartTime: 14:37:28.185
Connection: close
```

REST API Interaction Tools

REST API Interaction Tools

- **Beyond the browser, there are a number of tools that may be used to interact with REST API's**
  - cURL
  - Swagger
  - wget
  - Postman
- **This short course will focus on using:**
  - cURL
  - Swagger

cURL Overview

- **cURL is a command line tool for getting or sending files using URL syntax**
- **Supports a range of common Internet protocols, including:**
  - HTTP
  - HTTPS
  - FTP
  - FTPS
  - LDAP
  - RTSP
  - Etc…
- **cURL supports HTTPS and performs SSL certificate verification by default when a secure protocol is specified such as HTTPS**
- **cURL is free and open source software**
- **Available from https://curl.haxx.se/ or through your package manager**

cURL commandline syntax

- **cURL supports a command line syntax common with Linux based utilities**

SYNTAX

```
curl [options...] <url>
```

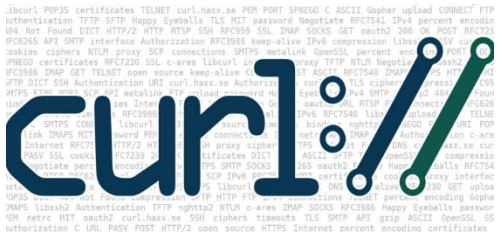
Or

```
curl <url> [options...]
```

Example

```
curl www.Ciena.com
curl -v www.Ciena.com
curl www.Ciena.com -v
```

Notable cURL command line switches



Switch	Description
-i, --include	Include protocol response headers in the output
-k, --insecure	Allow insecure server connections when using SSL
-H,--header <header>	Extra header to use when getting a web page You may specify any number of extra headers
-X, --request <command>	(HTTP) Specifies a custom request method to use when communicating with the HTTP server The specified request method will be used instead of the method otherwise used (which defaults to GET) Read the HTTP 1.1 specification for details and explanations Common additional HTTP requests include POST, PUT, and DELETE

```
Example:
curl -X GET --header "Accept: application/json"
"https://10.41.89.52/bpocore/market/api/v1/products?includeInactive=false&offset=0&limit=1000"
```

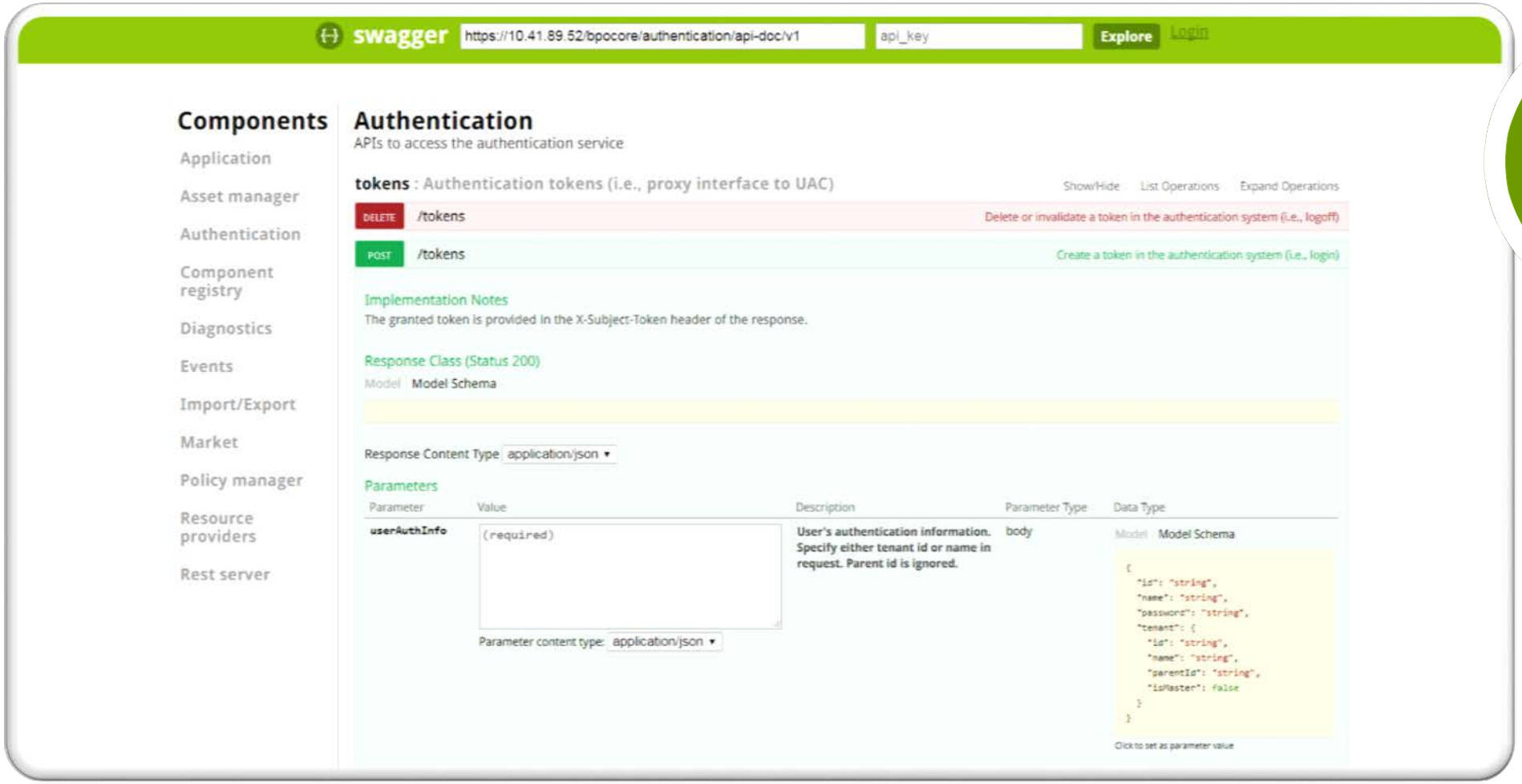
Swagger Overview

- **Swagger is a set of open-source tools built around the OpenAPI Specification that can help you interact with RE**
  - Design
  - Build
  - Document
  - Consume
- **Uses JavaScript Object Notation (JSON), or YAML**
- **Major Swagger tools include:**
  - Swagger Editor
  - Swagger UI
  - Swagger Codegen
- **OpenAPI Specification (formerly Swagger Specification) is an API description format for REST API's**
  - An OpenAPI file allows you to describe your entire API

Swagger UI

- **The Swagger UI allows anyone to interact and visualize API resources without having to worr any implementation logic**
- **The UI is automatically generated from the Swagger specification**
  - Includes visual documentation
  - Trivial for client side consumption of information
- **Ships as part of the DevOps toolkit**
- **Accessible at:**
  - `https://[serveraddress]/bpocore/swagger#`

Swagger UI Screenshot



Displaying Blue Planet API's in Swagger

- **Developers may display and call all Blue Planet APIs using the Swagger interface**
- **Swagger allows you to interact with the Blue Planet APIs using graphical elements so devel learn how the APIs respond to different parameters and options**
  - Also allows you to initiate API's on the Blue Planet host
- **For each API, Swagger provides the Blue Planet API:**
  - Schema
  - Supported operations
  - Parameters
  - Required authorization
  - Response messages
  - Request URL
  - Request curl command
  - Command response

Task 01

- **Task 01 is optional**
- **Perform task 01 now if you do not have cURL installed**

API Authentication

Authentication API

- **The Authentication component is responsible for authenticating users with the Blue Planet User Access Control system**
  - Known as Tron
- **The User Access Control system manages:**
  - Tenants
  - Users
  - Roles
    - Role management is per user, per application
- **The Authentication API affords administrators and developers the ability to:**
  - Verify that the application is properly connected
  - Obtain permission tokens used in authenticated API calls
  - Release permission tokens used in authenticated API calls

API Authentication

- **There are four methods to handle authentication of REST API requests from clients**
  - Hashed Message Authentication Code (HMAC) signed requests
    - Uses HMAC signatures in the HTTP request header
    - Recommended for production environments.
  - Token authentication
    - Using a short-lived token without message signing for user-based authentication
  - Swagger UI
    - Authenticates users through the Blue Planet User Access Control (UAC)
  - No authentication
    - Runs the Planet Orchestrate application with no API request authentication
    - Only use in development environments

HMAC Signed Requests

- **In production, REST clients of Planet Orchestrate should authenticate by including a Hashed Message Authentication Code (HMAC) signature in the Authorization header of each HTTP request**
  - A key pair (key identifier and key secret) must be obtained from the Blue Planet User Access Control system (UAC)
  - The key secret is used to compute the MAC signature
  - The header value is a string that starts with the "MAC" and then contains the key id, timestamp, nonce, and computed MAC

VALUE	IDENTIFIER	DESCRIPTION	EXAMPLE
Key Identifier	id	String identifying the key pair	**id="ae71d7d92d7d4c659a7d3336db6c4c99"**
Timestamp	ts	Number of elapsed seconds from UNIX Epoch	**ts="1411065195"**
Nonce	nonce	A one-time use string (preferably a URL-safe Base64 Encoding per RFC 4648)	**nonce="Jw1ctgzz2X2n+6DDOBl Eig=="**
MAC	mac	Signature	**mac = "oYhbGKDhOZZ9ReHQyZS0jMLwOSQDGplmWbtY3d+dORM="**

Token Authentication

- **Exchange user login credentials (user name, password, and tenant name) for a short-lived token (24 hour default)**
  - The token is included in the Authorization HTTP header and grants the caller access appropriate for the corresponding user roles
  - The response header value is in the format token `token:value`

Example:

```
curl -i -k -X POST --header "Content-Type: application/json" --header "Accept: application/json"
-d "{\"username\": \"849student\",\"password\": \"849password\",\"tenant\": \"master\"}"
"https://10.41.89.52/tron/api/v1/tokens"
```

Swagger Authentication Component Token Authentication

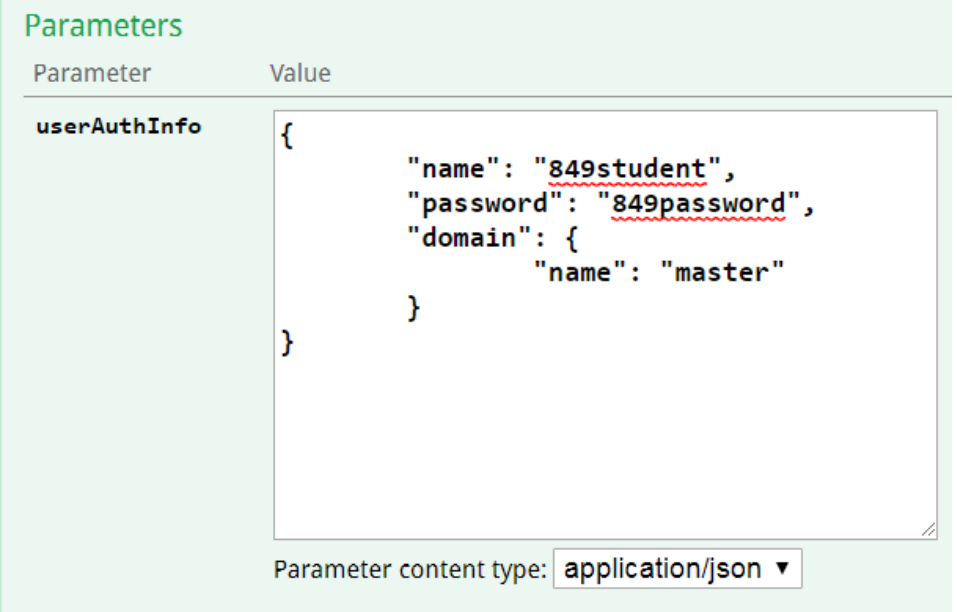
- **A token may also be retrieved from the Swagger Authentication Component**
  - **The syntax for the to request the token is slightly different than interacting with tron directly**
  - **The token is returned in the `X-Subject-Token` response header in the form of `X-Subject-Token: <token>`**
- **Users may also log into swagger**
  - **Once logged in, a Authorization: Bearer <token> header will be added to each one of the sample cURL calls**

Example: Retrieve Token

```
curl -i -k -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{\"name\": \"849student\",\"password\": \"849password\",\"domain\": {\"name\": \"master\"}}" "https://10.41.89.52/bpocore/authentication/api/v1/tokens"
```

Supplying A Template To The Swagger Authentication Component

- **A json formatted template containing the salient login information may be supplied to the Swagger Authentication Component directly**
  - A json template should be supplied to the `userAuthInfo` Field
  - An http 201 success code will be returned and the token will be the value of the "x-subject-token" field if the exchange is successful.



Using The Exchanged Token

- **Whether you obtain a token from the UAC or the Swagger Authentication component, the token is used the same way when making calls using cURL**
- **Generated tokens are added to the "`Authorization: token`" request header field of all subsequent commands**

Example: Use Token

```
curl -v -k -H "Content-Type:application/json" -H "Authorization: token
8bc1afd3d29d35a54ef8" -X GET
"https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000"
```

User Access Control Authentication

- **API calls initiated through the Blue Planet Swagger UI are validated through UAC**
  - A Bearer authentication string is added to the Authorization header by Swagger if the user is logged in with the appropriate credentials

Example:

```
curl -X GET --header "Accept: application/json" --header "Authorization: Bearer 8c3329dbf25c8f9ff3d8"
"https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000"
```

No Authentication

- **Blue Planet Orchestrate can run in no-auth (no authentication) mode to allow API calls to run without authentication**
  - For testing and development purposes only
- **Running Orchestrate in no-auth mode gets around 401 errors caused by lack of authentication**
- **Developers lose the ability to establish a true user identity, tenant, or roles which might impact other functionality**
- **Best practices dictate that the development environment emulate the production environment as much as possible**

Task 02

- **Perform task 02 now**

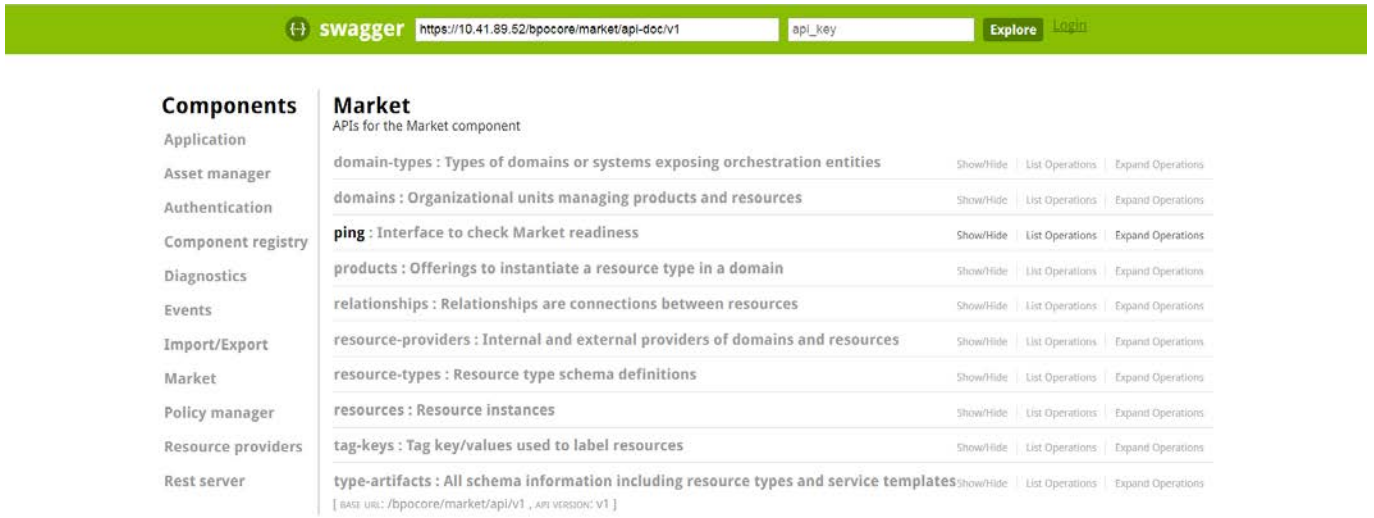
Demo - Token Based Authentication Walkthrough

- **This walkthrough will demonstrate how to use cURL to return an access token that may be used for successive API queries**
- **The walkthrough will also demonstrate how to yield the same information from the Swagger interface**

Core API’s

Blue Planet Orchestrator Core API's

- **The Blue Planet Orchestrator core API's are organized into logical groups known as components**
- **Authentication API's have been previously discussed**
- **Market API's will be discussed in a subsequent section**
- **This section will focus on the rest of the Core API's**



Other Planet Orchestrate Components

- **Beyond the Core API's, Blue Planet Orchestrator supports a number of other component API's**
  - Application
  - Asset Manager
  - Component Registry
  - Diagnostics
  - Events
  - Import/Export
  - Policy Manager
  - Resource Provider
  - Rest-Server

Application, Asset, and Component Registry Components

- **Application**
  - The Application component exposes management and configuration API's for Planet Orchestrate
- **Asset Manager**
  - The Asset Manager manages file assets used by the system and is built on top of a git repository:
  - Manages:
    - Resource types
    - Service templates
    - UI Schemas
- **Component Registry**
  - Maintains a registry of active system components
    - System components may further identify themselves upon registration by using `tagKey, tagValue` pairs
  - Reg/UnRegisters components
  - Lists, queries, or gets registered components
  - Manages components

Diagnostics, and Import/Export Components

- **Diagnostics**
  - Exposes an interface to define, run, and view self-diagnostics tests
    - Some are used by the Planet Orchestrate daemon to run tests that are reported back to the Nagios monitoring system
    - Two resources exposed are:
      - Testcases
      - Test-Instances
- **Events**
  - Provides an asynchronous message bus to publish event information
  - Exposes the following API's
    - Get or List Topics
    - Create and event topic
- **Import/Export**
  - This component is used to importing and exporting JSON data
    - Supports a number of JSON Entities
    - Importing large amounts of JSON data
    - Importing into production clusters
    - Exports behaves similar to imports, except data is sent from the system

Policy Manager, Resource Provider, and Rest-Server Components

- **Policy Manager**
  - Responsible for implementing authorization and other policies
  - The API exposes the following resources
    - Realms
    - Conditions
    - Policies
- **Resource Provider**
  - Exposes API's to access information about the resource providers that has been availed to the system
  - The component exposes a single top-level resource that defines the following sub-resources
    - Product
    - Resource
    - Relationship
- **Rest-Server**
  - Coordinating hub for all of the other components that expose RESTful API's
    - Exposes and API to list all other components
      - API may be used to seed API discovery for each system

Task 03

- **Perform Task 03 now**

Working With The Market API

Market API's

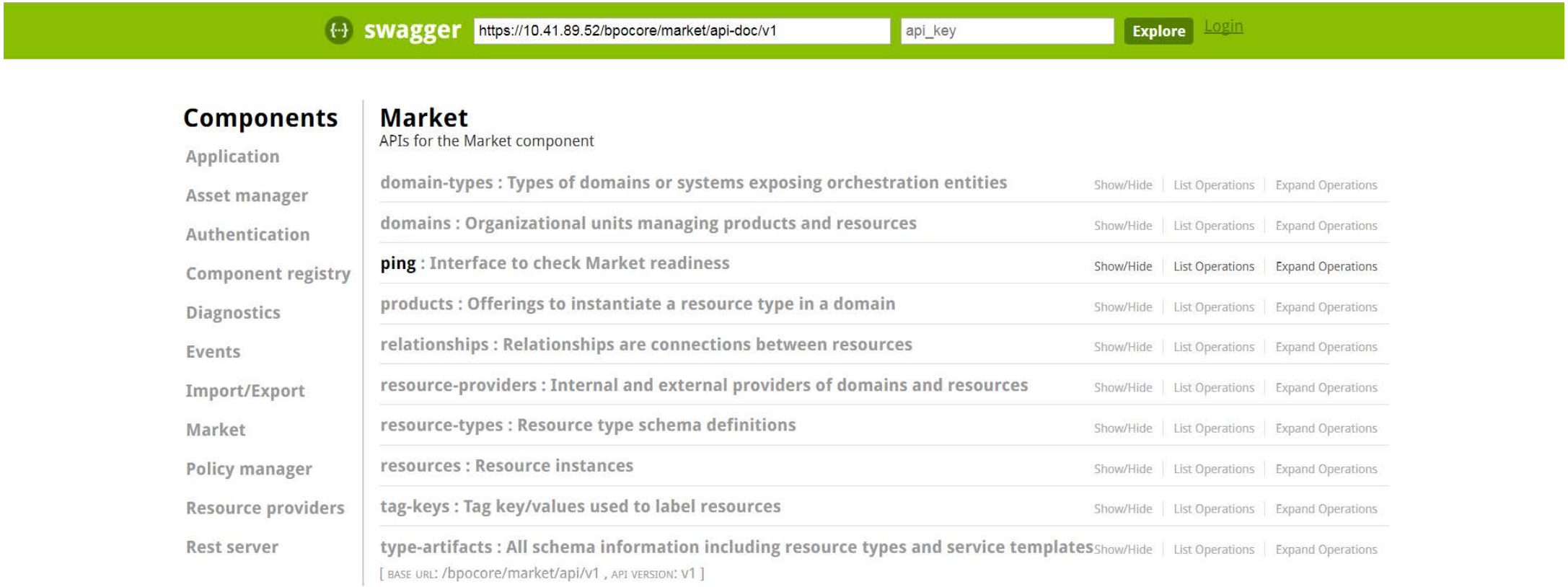
- **The Market component is the primary component for defining, creating, and managing resources in the domains**
- **Developers will use the following Market component APIs to define, create, and manage resources in the orchestration domains**
  - Typically use these APIs more frequently than other Blue Planet component APIs.

API	Description	Methods
domain-type	A type of domain or system that identifies the orchestration Entities	GET
domain	A domain-type instance specified with account credentials and managed by the orchestrator	GET, POST,DELETE, PATCH, PUT
resource-type	A resource type instance managed by Blue Planet Orchestrate	GET
resource	A resource instance that is requested or created	GET, POST,DELETE, PATCH, PUT

Market API's (con't)

API	Description	Methods
product	Links resources to domains Each resource is linked to one product Each product is linked to one domain Stored in the product catalog	GET, POST, DELETE, PATCH, PUT
relationship	Defines the associations among resources.	GET, POST, DELETE
type-artifact	Defines the meta-information for all type information	GET, POST, DELETE, PATCH, PUT
tag-key	Arbitrary tag-key, tag-value pair that can be associated with a resource to facilitate filtering.	GET, POST, DELETE, PATCH, PUT
ping	Pings the component to see whether it is ready to handle requests.	GET

Demo – Using the Market API's Through Swagger And Curl Walkthrough



cURL – Market Examples

- **Examples:**
  - domain
    - curl -X GET --header "Accept: application/json" --header "Authorization: token c903158fd364d94bf06b" https://10.41.89.52/bpocore/market/api/v1/domains/59777a39-b4bd-444b-9304-6e01cd3f0c58
  - resource-types
    - curl -X GET --header "Accept: application/json" --header "Authorization: token c903158fd364d94bf06b" https://10.41.89.52/bpocore/market/api/v1/resource-types?offset=0&limit=1000
  - resources
    - curl -X GET --header "Accept: application/json" --header "Authorization: token c903158fd364d94bf06b" https://10.41.89.52/bpocore/market/api/v1/resources?offset=0&limit=1000
  - products
    - curl -X GET --header "Accept: application/json" --header "Authorization: token c903158fd364d94bf06b" "https://10.41.89.52/bpocore/market/api/v1/products?includeInactive=false&offset=0&limit=1000"

Filtering Results

- **GET requests sent to Blue Planet Orchestrator typically return lists of items**
  - These result sets may be large
- **The Blue Planet Orchestrator API's give developers the ability to filter the results returned**
  - Filtering types supported are:
    - Exact Match Query Filter
    - Partial String Match Query Filter
    - String Matching Syntax
    - Tag Queries
    - Errors

Exact Match

- **The exact-match query filter returns only results that match a specified string**
- **The exact-match query filter has the following syntax**
  - Filters are query parameters specified by the parameter name `q`
  - Each fitter is a key/value pair separated by a colon (`key:value`)
  - And conditions may be specified by using a comma (,) to separate groups of key value pairs
  - Parenthesis around the whole value are optional
- **Example:**
  - `curl -i -k -X GET --header "Accept: application/json" --header "Authorization: token d3936103133d34e4ab10" "https://10.41.89.52/bpocore/market/api/v1/products?includeInactive=false&q=(resourceTypeId:tosca.resourceTypes.NumberPool) "`
  - `curl -i -k -X GET --header "Accept: application/json" --header "Authorization: token d3936103133d34e4ab10" "https://10.41.89.52/bpocore/market/api/v1/resources?q=(label:vCPE8675309, discovered:false,orchState:active)"`

Complete The Rest Of The Tasks In The Lab Guide

- **Complete tasks 04 – 06 now**

Questions?

Thank You for Attending