Automation

Dynatrace Training Module

Agenda

- Environment API
  - General Info
  - Authentication
  - API Explorer
  - Query Dynatrace Data
  - Enhance Dynatrace Data
  - Configuration
- Cluster Management API
- Command Line Interface (CLI)
- Auto-remediation
- Load Testing Integration

Environment API

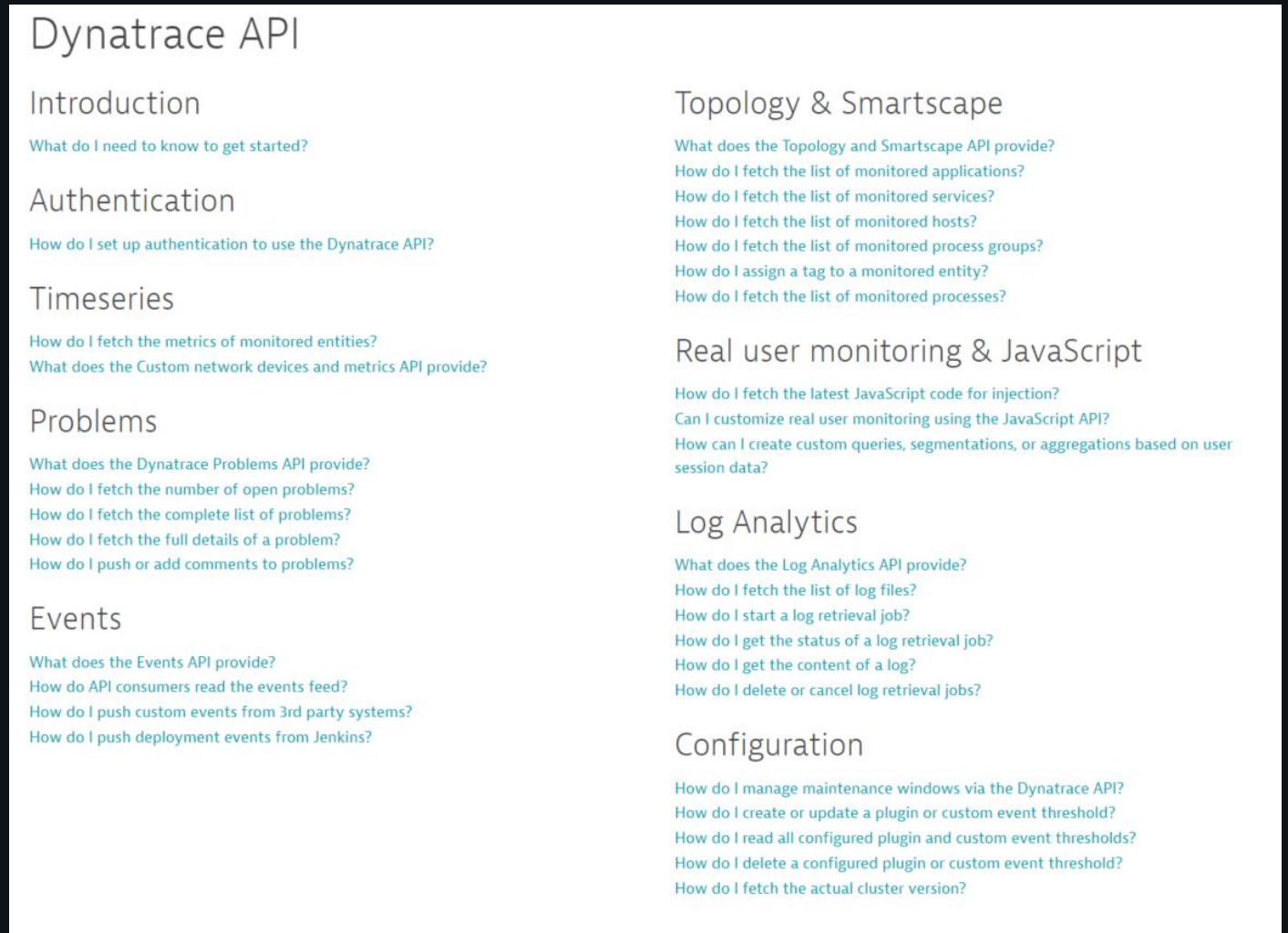
General Info

Open APIs Are the New Open Source

- Dynatrace emphasizes powerful APIs
- Large scale deployments need APIs to cope with automated configuration
- DevOps love APIs to build their automated tasks around
- We offer API openness, in order to integrate with any other service or platform
- This allows Dynatrace to position itself as a monitoring Hub within existing software ecosystems and simplify integration with legacy systems

API Documentation

- https://www.dynatrace.com/support/help/dynatrace-api/



Dynatrace API

Introduction

What do I need to know to get started?

Authentication

How do I set up authentication to use the Dynatrace API?

Timeseries

How do I fetch the metrics of monitored entities?
What does the Custom network devices and metrics API provide?

Problems

What does the Dynatrace Problems API provide?
How do I fetch the number of open problems?
How do I fetch the complete list of problems?
How do I fetch the full details of a problem?
How do I push or add comments to problems?

Events

What does the Events API provide?
How do API consumers read the events feed?
How do I push custom events from 3rd party systems?
How do I push deployment events from Jenkins?

Topology & Smartscape

What does the Topology and Smartscape API provide?
How do I fetch the list of monitored applications?
How do I fetch the list of monitored services?
How do I fetch the list of monitored hosts?
How do I fetch the list of monitored process groups?
How do I assign a tag to a monitored entity?
How do I fetch the list of monitored processes?

Real user monitoring & JavaScript

How do I fetch the latest JavaScript code for injection?
Can I customize real user monitoring using the JavaScript API?
How can I create custom queries, segmentations, or aggregations based on user session data?

Log Analytics

What does the Log Analytics API provide?
How do I fetch the list of log files?
How do I start a log retrieval job?
How do I get the status of a log retrieval job?
How do I get the content of a log?
How do I delete or cancel log retrieval jobs?

Configuration

How do I manage maintenance windows via the Dynatrace API?
How do I create or update a plugin or custom event threshold?
How do I read all configured plugin and custom event thresholds?
How do I delete a configured plugin or custom event threshold?
How do I fetch the actual cluster version?

URL

- HTTPS
  - The use of a secured communication channel over HTTPS is mandatory
- Dynatrace SaaS
  - *https://{DynatraceURL}/api/v1/*
- Dynatrace Managed URL Prefix
  - *https://owndomain/e/{id}/api/v1/*

General Info

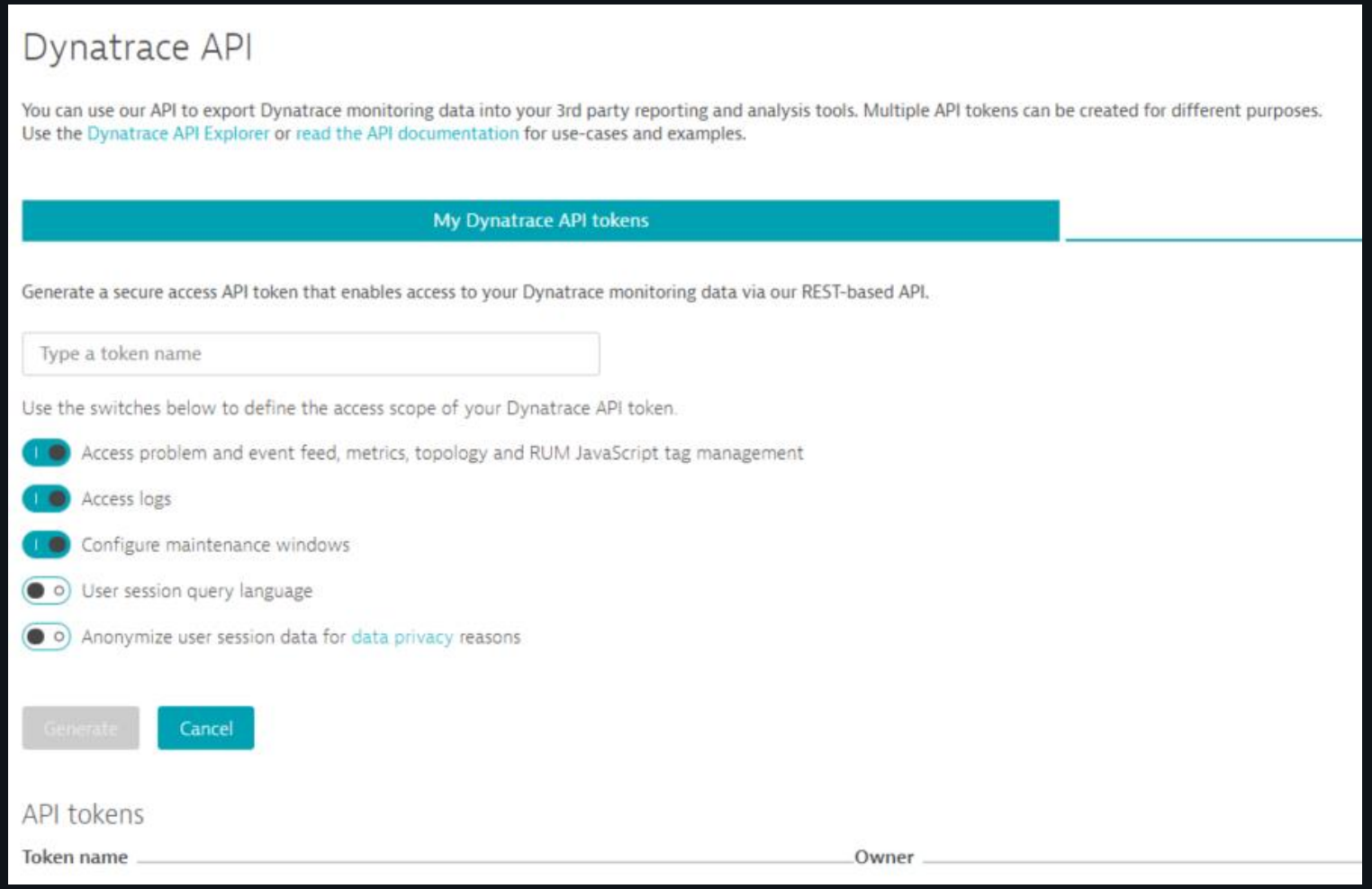
- Pricing
  - Read access to the Dynatrace API is free of charge on a fair use model
  - You are charged for defining and pushing new custom metrics through the Dynatrace API on a per-metric, per-month basis
- Rate limiting
  - API access is limited to 50 requests / minute / API on Dynatrace SaaS environment
  - There is no rate limit for Dynatrace Managed installations
- Custom network device and custom metric limits
  - For testing and prototyping, you have a free tier of “100 + 10 x Host units” custom metrics per month quota for each Dynatrace SaaS environment or Dynatrace Managed cluster
  - Each metric dimension counts as a separate metric

Authentication

Dynatrace API

- Authentication is achieved via a user-generated access key (available in your Dynatrace environment settings)
- Log into your Dynatrace environment and go to Settings > Integration > Dynatrace API
- Generate a new access token by typing a unique string into the Key label field, then click the Generate key button.

Dynatrace API Token



API Token Header

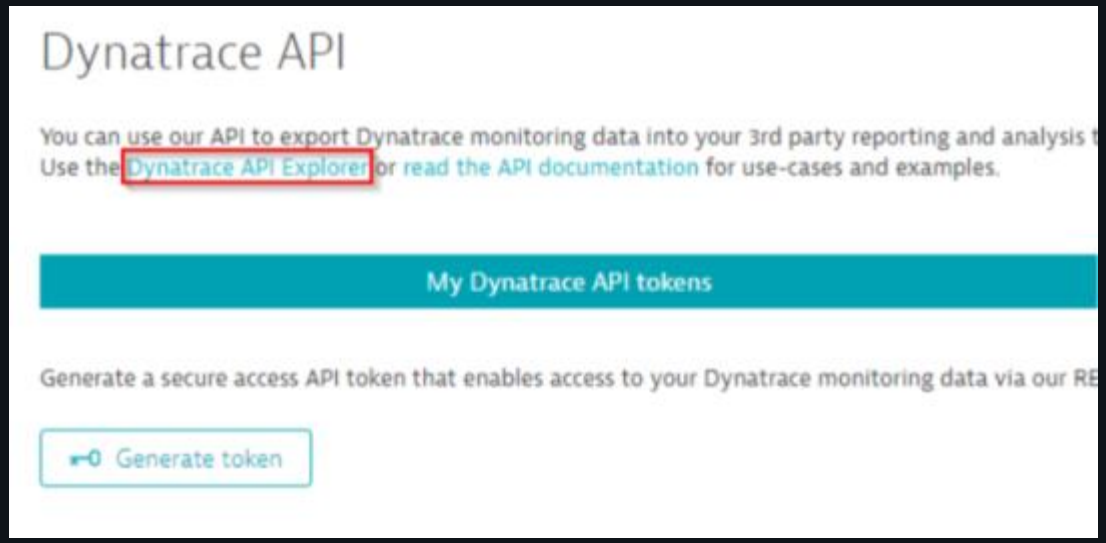
- Passing your API token within an authorization header
  - To authorize your API request using an HTTP header, use the previously generated Dynatrace token along with an 'Api-Token' realm.
    - *Authorization: Api-Token cw88t44BRk2KcJkdM419T*
  - The following example authenticates you with the Dynatrace API using the HTTP authorization header field.
    - *curl -L -H "Authorization: Api-Token cw88t44BRk2KcJkdM419T" "https://{DynatraceURL}/api/v1/timeseries"*

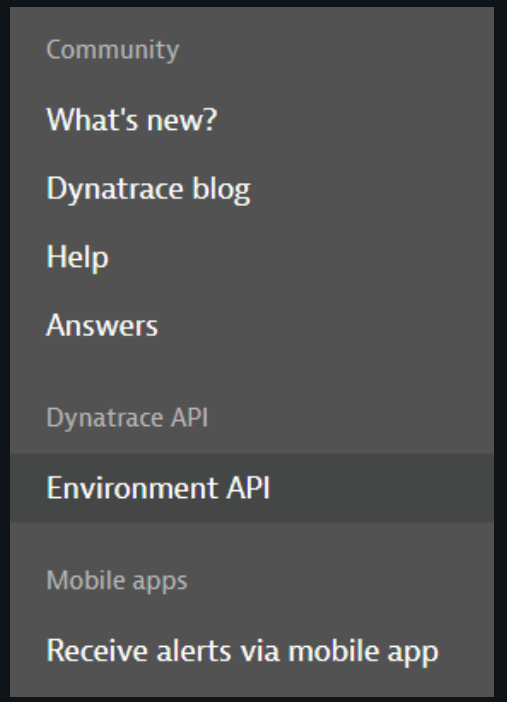
- Passing your API token within a query parameter
  - Authentication can optionally be achieved via a query parameter using your generated access key.
    - *curl -L "https://{DynatraceURL}/api/v1/timeseries?Api-Token=cw88t44BRk2KcJkdM419T"*

API Explorer

API Explorer

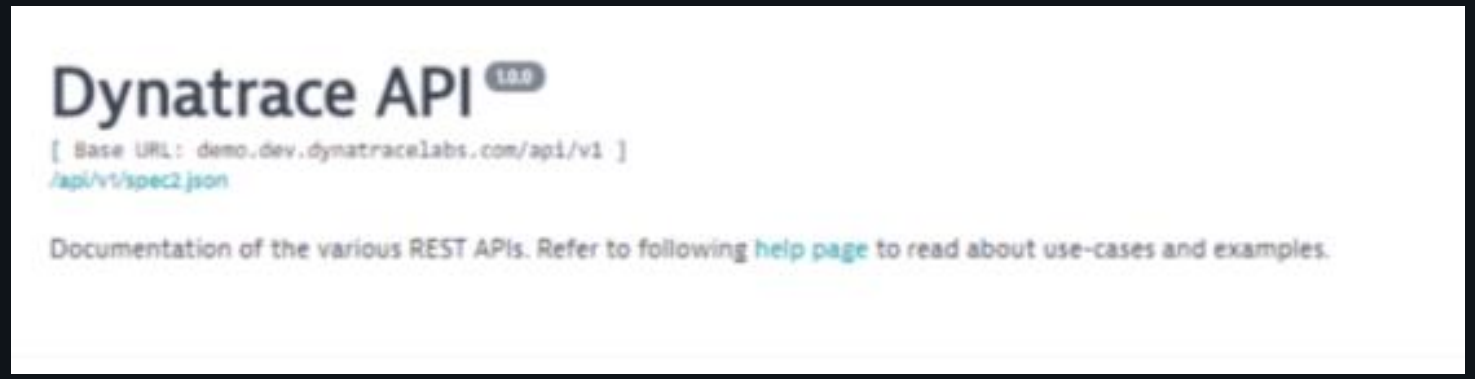
- API explorers allow you to review all existing endpoints and directly try out those endpoints
- Because an OpenAPI Specification (OAS) is now automatically included for each Dynatrace REST endpoint, you're assured that all recent changes are automatically reflected within the new Dynatrace API explorer





API Explorer

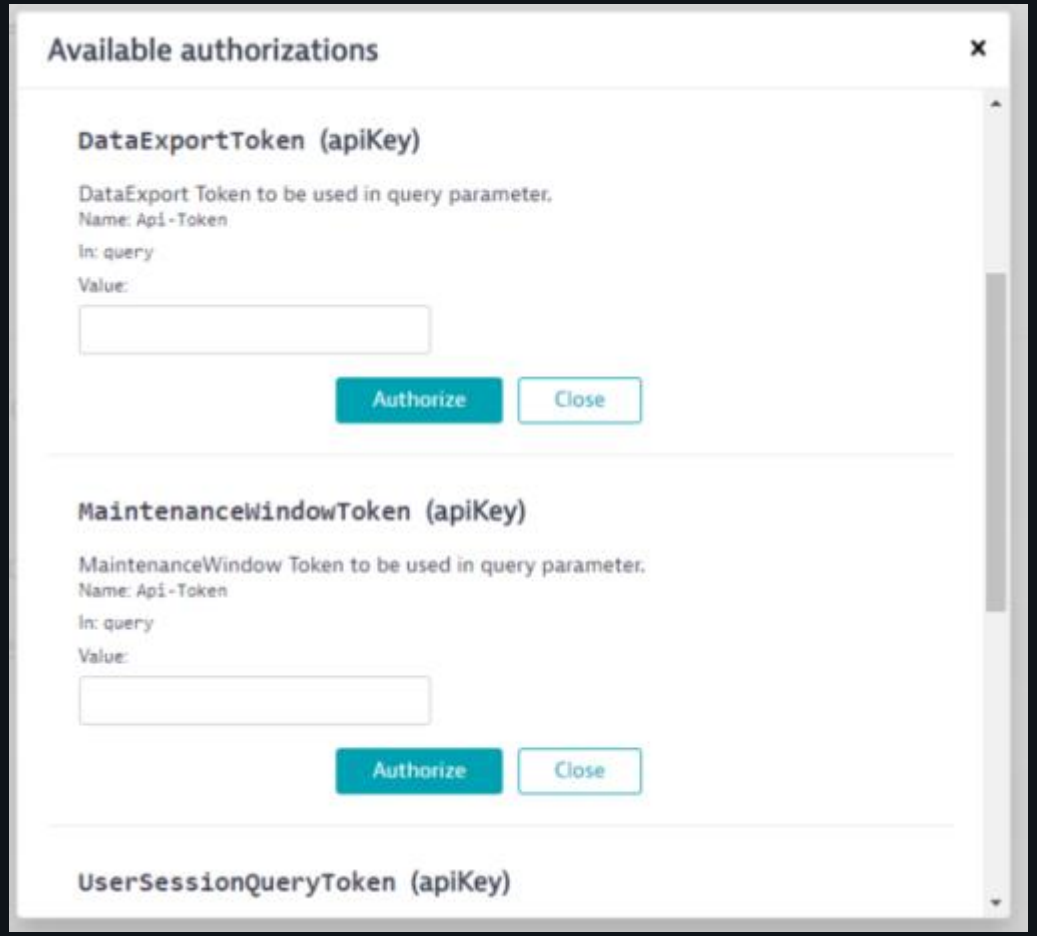
- The Dynatrace API explorer lists all available API endpoints for the given Dynatrace environment
- You'll recognize the traditional API endpoints for pulling time-series or topological information from your monitored environment
- Just below the title, you'll find a link to the raw OAS specification, which can be used with any OAS-compatible tool (for example, Swagger)



API Explorer

API Explorer

- When you open one of the given API endpoints, a dialog appears with information about the API tokens that secure the endpoint you've selected
- By entering your personal API token into the global Available authorizations dialog, you unlock all related API endpoints
- Once you've entered your API token, you can directly execute API calls within the API explorer



Cluster Version

GET /config/clusterversion Gets the current version of the cluster server.

Parameters Cancel

No parameters

Execute Clear

Responses Response content type application/json

Curl

```
curl -X GET "https://[redacted].com/api/v1/config/clusterversion?Api-Token=[redacted]" -H "accept: application/json"
```

Request URL

```
https:/[redacted]api/v1/config/clusterversion?Api-Token=[redacted]
```

Server response

Code	Details
200	Response body

```
{
  "version": "1.148.0.20180606-193059"
}
```

Response headers

```
content-encoding: gzip
content-length: 57
content-type: application/json
date: Thu, 07 Jun 2018 07:07:35 GMT
server: ruxit server
strict-transport-security: max-age=31536000;includeSubDomains
vary: Accept-Encoding
x-oneagent-js-injection: true
x-ratelimit-limit: 50
x-ratelimit-remaining: 49
x-ratelimit-reset: 1528355315633000
x-robots-tag: noindex
```

Responses

Code	Description
200	*successful operation*

Query Dynatrace Data

Timeseries API

Timeseries API

- *https://{DynatraceURL}/api/v1/timeseries*

- The timeseries endpoint delivers metrics that Dynatrace collects from the different monitored entities over time
- This endpoint is used to read metrics, such as CPU usage, for selected entities over a given timeframe
- By passing additional parameters this endpoint allows you to filter the selected timeseries for entity types and to specify what type of result aggregation the result should contain.

Timeseries API

- The timeseries endpoint allows the following parameters to be sent as HTTP GET requests
  - **timeseriesId** - Case-sensitive identifier for the specific timeseries that should be returned
    - For example, com.dynatrace.builtin:host.cpu.load
  - **startTimestamp** (optional) - Start timestamp of the requested timeframe, given in milliseconds (UTC)
  - **endTimestamp** (optional) - End timestamp of the requested timeframe, given in milliseconds (UTC)
  - **relativeTime** (optional) - It is possible to specify 'hour' to receive the timeseries data for the last hour
    - Allowed values are 'hour', '2hours', '6hours', 'day', 'week' and 'month';
  - **queryMode** (optional) - Defines the type of result that the call should return
    - Valid result modes are 'series' for receiving the entire timeseries as data points or 'total' for receiving one scalar value for the requested timeframe
  - **aggregationType** - Defines which aggregation type is used for the resulting timeseries
    - Valid aggregation types are min, max, avg, sum, median, and count
  - **entity** (optional) - Defines which entity should deliver the requested timeseries
    - You can find the specific entity ID by opening the corresponding entity page where the unique ID is displayed within the URL

Result

- Query results are returned as JSON payloads

- The result payload also contains attributes describing:
  - The unit (unit) of the returned data points
  - The result resolution (resolutionInMillisUTC) of the series
  - The selected type of aggregation (aggregationType)


```
{
"result":
  {
  "dataPoints":
    {
    "HOST-B64B6B9CB11E2244":
    [
    [1443418260000,36.186815897623696],
    [1443418320000,31.984347025553387],
    [1443418380000,36.897412618001304],
    [1443418440000,33.65802764892578],
    ...
    ]
    },
  "timeseriesId":"com.dynatrace.builtin:host.cpu.user",
  "unit":"Percent",
  "entities":{"HOST-B64B6B9CB11E2244":"L-W8-64-APMDay3"},
  "resolutionInMillisUTC":60000,
  "aggregationType":"AVG"
  }
}
```

Example

- In this example the request reads the result of the Apdex index (com.dynatrace.builtin:app.apdex metric) for all applications within the last day
- The result returns Apdex as count for each application

Response content


```
{"result": {
   "dataPoints":    {
      "APPLICATION-8BC2DDAD": [      [
         1517906880000,
         0
      ]],
      "APPLICATION-EA7C4B59": [      [
         1517906880000,
         1
      ]],
      "APPLICATION-CA9F9519": [      [
         1517906880000,
         0.5
      ]]
   },
   "timeseriesId": "com.dynatrace.builtin:app.apdex",
   "unit": "PerMinute (count/min)",
   "entities":    {
      "APPLICATION-8BC2DDAD": "My internal portal",
      "APPLICATION-CA9F9519": "My mobile application",
      "APPLICATION-EA7C4B59": "My web application"
   },
   "resolutionInMillisUTC": 86400000,
   "aggregationType": "COUNT"
}}
```

Example

- In this example the request returns total values of received bytes (com.dynatrace.builtin:host.nic.bytesreceived metric) for the HOST-B66B773, HOST-600F77E5, and HOST-A07C6666 entities within the last day
- The result returns the total averages for the network interfaces, related to specified hosts

Response content

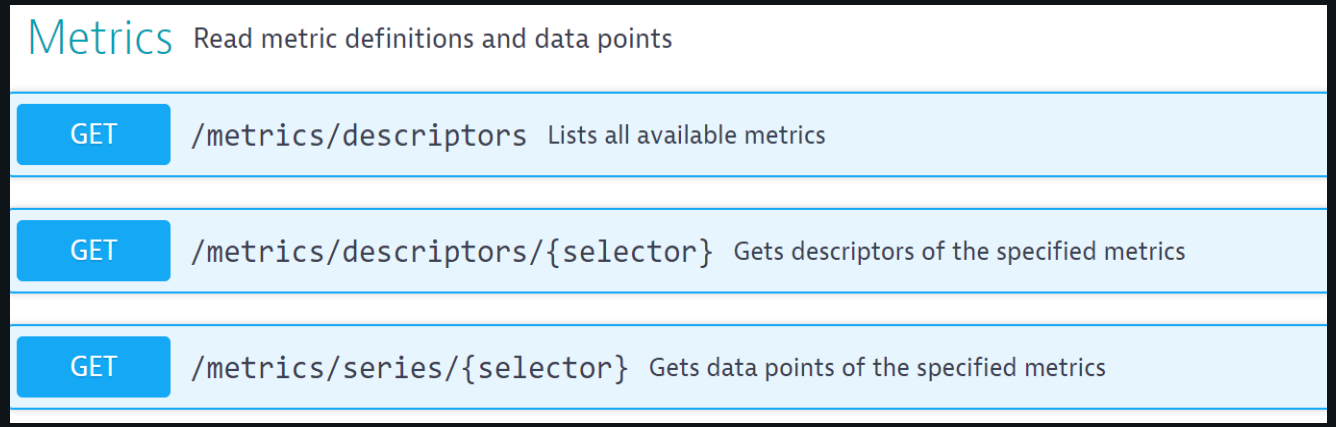

```
{"result": {
   "dataPoints":    {
      "HOST-600F77E5, NETWORK_INTERFACE-600F77B5": [      [
         1517925540000,
         15073.498178864322
      ]],
      "HOST-B66B773D, NETWORK_INTERFACE-B66445C84BE": [      [
         1517925540000,
         274277.79485396383
      ]],
      "HOST-A07C6666, NETWORK_INTERFACE-88E0C76F2D": [      [
         1517925540000,
         1226019.726008345
      ]]
   },
   "timeseriesId": "com.dynatrace.builtin:host.nic.bytesreceived",
   "unit": "BytePerSecond (B/s)",
   "entities":    {
      "NETWORK_INTERFACE-88E0C76F2D": "eth0",
      "NETWORK_INTERFACE-B66445C84BE": "vmxnet3 Ethernet Adapter",
      "HOST-B66B773D": "l-02",
      "HOST-600F77E5": "G57",
      "NETWORK_INTERFACE-600F77B5": "Intel(R) PRO/1000 MT Network Connection",
      "HOST-A07C6666": "l-03"
   },
   "resolutionInMillisUTC": 86400000,
   "aggregationType": "AVG"
}}
```

Some examples

- Fetching a metric/list of metrics
  - https://demo.live.dynatrace.com/api/v1/timeseries/?Api-Token=<YOUR_TOKEN>
  - https://demo.live.dynatrace.com/api/v1/timeseries/?filter=plugin&Api-Token=<YOUR_TOKEN>
  - https://demo.live.dynatrace.com/api/v1/timeseries/?relativeTime=hour&aggregationType=AVG&timeseriesId=com.dynatrace.builtin:host.cpu.user&entity=HOST-6D01CC4CCDA915AA&Api-Token=<YOUR_TOKEN>
  - https://demo.live.dynatrace.com/api/v1/timeseries/?relativeTime=hour&aggregationType=AVG&timeseriesId=com.dynatrace.builtin:host.cpu.user&Api-Token=<Api-Token>

Environment API v2 (Metrics)

- The new version offers a lot more convenience in building metric queries and it allows you to efficiently filter huge result sets.
- New Metric selector, ability to select 1 or more metrics, available csv format, percentiles, convenient time selector, and flexible filtering for relevant results.



Problems API

Problems API

- *https://{DynatraceURL}/api/v1/problem/*

- This family of endpoints delivers metrics and details about problems that Dynatrace detects within a given environment
- The returned list of problems is identical to that shown in the Dynatrace Web UI and within the Dynatrace mobile app
- A single problem typically contains summary information, impact analysis, and a list of any events that are correlated with the problem

Problems API

- The Dynatrace Problems API follows a particular strategy
  - First, high-level status information is returned by calling the *status* endpoint
  - Then the current problem feed is fetched (minimal problem details included) by calling the *feed* endpoint
  - Once a problem is identified, the API calls the *details* endpoint to fetch all detailed information related to the problem
    - Note that problem details are often large in size.

Status Endpoint

- *https://{DynatraceURL}/api/v1/problem/status*

- A call to the /problem/status endpoint returns the number of currently open problems organized by impact level (i.e., affecting applications, services, or infrastructure) so you know immediately when problems affect your customers.

- Result
  - totalOpenProblemsCount: Number of open problems in your environment.
  - openProblemCounts: Number of open problems in your environment, organized by impact level (application, services, and/or infrastructure).

```
result: {
  totalOpenProblemsCount: 4,
  openProblemCounts: {
    APPLICATION: 1,
    INFRASTRUCTURE: 3,
    SERVICE: 0
```

Feed Endpoint

- *https://{DynatraceURL}/api/v1/problem/feed*

- A call to /problem/feed returns the list of problems observed by Dynatrace during a relative period of time
- Complete problem details may be large because the feed contains a subset of each problems' meta information to give the caller the option of selecting all detail from specific problems
- The /problem/feed endpoint offers additional query and filter parameters to enable you to specify which types of problems the API consumer is interested in


```
{
  result: {
    problems: [
      {
        id: "1328992236593193841",
        startTime: 1453166820000,
        endTime: -1,
        displayName: "851",
        impactLevel: "INFRASTRUCTURE",
        status: "OPEN",
        rankedImpacts: [
          {
              entityId: "HYPERVISOR-CC7586844F686D21",
              entityName: "emea-gdn-vh026.emea.cpwr.corp",
              impactLevel: "INFRASTRUCTURE",
              eventType: "CPU_SATURATED"
          }
        ],
        affectedCounts: {
          APPLICATION: 0,
          INFRASTRUCTURE: 1,
          SERVICE: 0
        },
        recoveredCounts: {
          APPLICATION: 0,
          INFRASTRUCTURE: 0,
          SERVICE: 0
        }
      },
      ...
    ],
    monitored: {
      INFRASTRUCTURE: 1599,
      SERVICE: 56,
      APPLICATION: 10
```

Feed Endpoint

- The feed endpoint allows the following parameters to be sent as HTTP GET requests:
  - **status** (optional) Possible values: OPEN or CLOSED
    - Filters the resulting set of problems according to their status (OPEN or CLOSED)
    - Without status filtering the call returns all problems for the given time period.
  - **impactLevel** (optional) Possible values: APPLICATION, SERVICE or INFRASTRUCTURE
    - Filters the resulting set of problems based on impact level: applications (APPLICATION), services (SERVICE), or infrastructure component (INFRASTRUCTURE)
    - Without impactLevel filtering, the call returns all problems for the given time period
  - **relativeTime** (optional) Possible values: hour, 2hours, 6hours, day, week, month
    - Specifies the relative time period for receiving the problem feed
    - Without specifying this parameter, the call returns all problems observed within the last hour

Details Endpoint

- *https://{DynatraceURL}/api/v1/problem/details/{problemid}*

- To fetch complete meta information for a specific problem, the API consumer must call the endpoint /problems/details/{problemid}
- The placeholder {problemid} must be replaced with the actual unique problem ID
  - Problem IDs are found within the ID fields of each problem in the problem-feed JSON result
- The problem/details endpoint returns all problem meta information, including start time (startTime), end time (endTime, -1 if the problem is still open), the short problem number (displayName), impact level, and status
- Problem information also contains the array of events that were identified as being relevant to the given problem
- Each single event contains information about its impacted entity (entityId, entityName), the type of event, and the severity level of the given event.

Example

- The following example result shows an infrastructure problem that contains a CPU event on the given host


```
result: {
  id: "1328992236593193841",
  startTime: 1453166820000,
  endTime: -1,
  displayName: "851",
  impactLevel: "INFRASTRUCTURE",
  status: "OPEN",
  rankedEvents: [
    {
      startTime: 1453166820000,,
      endTime: 9223372036854776000,
      entityId: "HYPERVISOR-CC7586844F686D21",
      entityName: "emea-gdn-vh026.emea.cpwr.corp",
      impactLevel: "INFRASTRUCTURE",
      eventType: "CPU_SATURATED"
      status: "OPEN",
      severities: [
        {
          context: "CPU_USAGE",
          value: 96.42066955566406,
          unit: "%"
        },
        {
          context: "CPU_READY_TIME",
          value: 18.439342498779297,
          unit: "%"
```

Problem Comments Endpoint

- *https://{DynatraceURL}/api/v1/problem/details/{PID}/comments*

- You can share insights related to remediation actions or additional information about possible root causes by attaching comments to detected problems
- The problem comments REST endpoint enables third party integrations to push comments to existing problems or to share the collected feed of comments through the API

Example

- This example shows the result of a successful call to a problem's comments feed


```
{
  comments: [
    {
      id: -5785321710852868000,
      createdAtTimestamp: 1477561321162,
      content: "My UI comment. This is a known issue caused by a backup process run.",
      userName: "wolfgang@email.com",
      context: null
    },
    {
      id: -2656985228510593500,
      createdAtTimestamp: 1474543966351,
      content: "This is a comment with an [inline-style link](https://www.google.com).",
      userName: "Wolfgang",
      context: "Slack"
    }
```

Some examples

- Fetching problem feed and details
  - https://demo.live.dynatrace.com/api/v1/problem/feed/?relativeTime=week&Api-Token=<YOUR_TOKEN>
  - https://demo.live.dynatrace.com/api/v1/problem/details/5853544736714377710/?relativeTime=week&Api-Token=<YOUR_TOKEN>

Topology and Smartscape API

Topology and Smartscape API

- *https://{DynatraceURL}/api/v1/entity/*

- This family of endpoints delivers details about the applications, services, and infrastructure entities that Dynatrace automatically detects and monitors within a given environment
- The returned information contains important attributes about the monitored entities as well as outgoing and incoming relationships
- This family of endpoints is organized along the three major environment layers:
  - Applications
  - Services
  - Infrastructure

Applications Endpoint

- *https://{DynatraceURL}/api/v1/entity/applications*

- A call to the /applications endpoint returns a list of currently monitored applications along with their attributes and relationships.
- Result
  - **fromRelationships**: Outgoing connections from the application to other entities (e.g., calls to a service).
  - **toRelationships**: Incoming connections to the application.
  - **applicationType**: The application type (e.g., a RUM or synthetic application).
  - **tags**: The tags that have been defined for the application. This list contains all user-defined labels as well as all tags imported from third-party systems, such as AWS.


```
[
  {
    entityId: "APPLICATION-EA7C4B59F27D43EB",
    displayName: "RUM Default Application",
    customizedName: "RUM Default Application",
    tags: [
      {
        context: "USER",
        key: "mytest"
      },
      {
        context: "USER",
        key: "test"
      },
      {
        context: "USER",
        key: "anothertag"
      }
    ],
    fromRelationships: {
      calls: [
        "SERVICE-A6573F56394A4D90",
        "SERVICE-CF188BBF8568DE91",
        "SERVICE-E5AA28EFED7D593E"
      ]
    },
    toRelationships: { },
    applicationType: "DEFAULT"
  }
]
```

Services Endpoint

- *https://{DynatraceURL}/api/v1/entity/services*

- A call to the /services endpoint returns a list of currently monitored services along with their attributes and relationships.
- Result
  - fromRelationships: Outgoing connections from the service to other entities
  - toRelationships: Incoming onnections to the service from other entities.
  - agentTechnologyType: Shows the monitored technology type, such as Java or .NET.
  - serviceTechnologyType: Shows the technologies that the service is built on (e.g., Tomcat and Java).
  - tags: Returns the list of tags that have been defined for the service
  - serviceType: Shows the service type, such as web service, messaging service, or service method.


```
[
  {
    entityId: "SERVICE-713DAEE6611DE663",
    displayName: "JourneyService",
    tags: [
      {
        context: "USER",
        key: "Production"
      }
    ],
    fromRelationships: {
      calls: [
        "SERVICE-CA23CAF4CAAE890A",
        "SERVICE-B0ECCA259E8568E0"
      ]
    },
    toRelationships: {
      calls: [
        "SERVICE-2489BAD30B017B36",
        "SERVICE-30F0B00E7B2DF49A"
      ]
    },
    agentTechnologyType: "JAVA",
    serviceTechnologyTypes: [
      "Tomcat",
      "Java"
    ],
    serviceType: "WEBSERVICE",
    webServiceName: "JourneyService",
    webServiceNamespace: "http://webservice.busine
  }
]
```

Infrastructure Endpoint - Hosts

- *https://{DynatraceURL}/api/v1/entity/infrastructure/hosts*

- A call to the /infrastructure/hosts endpoint returns a list of currently monitored hosts along with their attributes and relationships.
- Result
  - fromRelationships: Outgoing connections from the host to other entities.
  - toRelationships: Incoming connections to the host.
  - tags: Returns the list of tags that have been defined for the host
  - osType: Shows the operating system type of the host, such as Linux or Windows.
  - osVersion: Shows the operating system version of the host

Infrastructure Endpoint – Process Groups

- *https://{DynatraceURL}/api/v1/entity/infrastructure/process-groups*

- A call to the /infrastructure/process-groups endpoint returns a list of currently monitored process-groups along with their attributes and relationships.
- Result
  - fromRelationships: Outgoing connections from the process-group to other entities.
  - toRelationships: Incoming relations to the process-group.
  - tags: Returns the list of tags that have been defined for the host
  - metadata: Shows available metadata of the process-group.
  - softwareTechnologies: Shows the software technologies that were detected for that specific process-group.

Entity API v2

Topology and Smartscape API

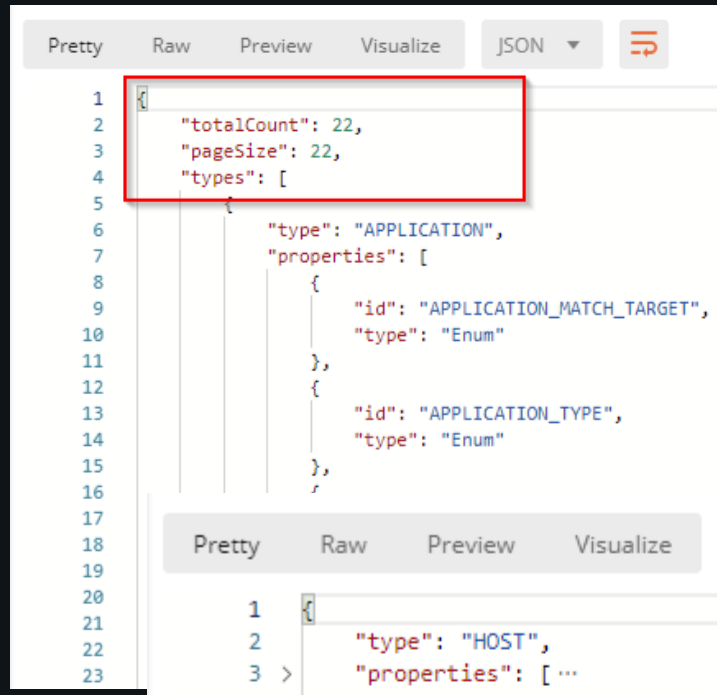
- *https://{DynatraceURL}/api/v2/entities*

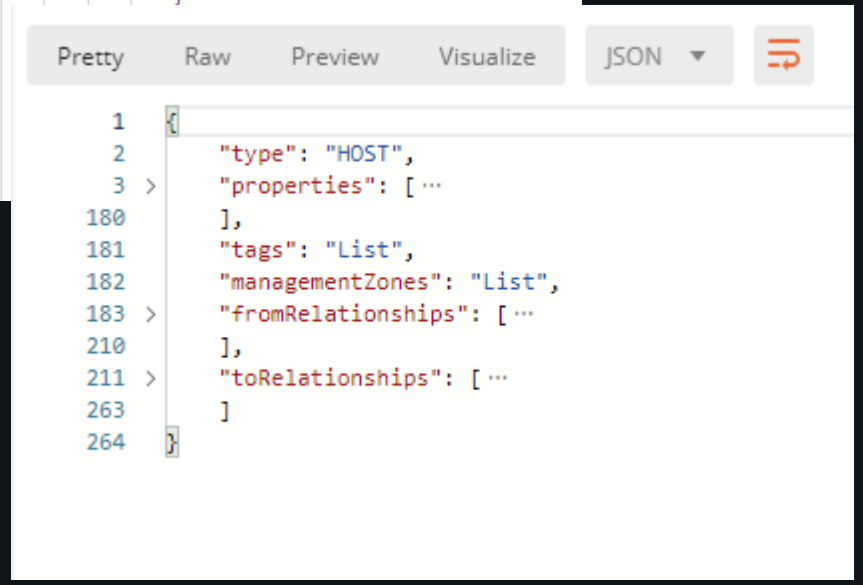
- This new API endpoint is specifically designed to enable efficient use of Smartscape information with external integrations. New features over v1 include:
  - One single endpoint to fetch information about any exported entity type
  - More entity types available for export
  - New API endpoint for getting the schema (type) of a given component type
  - Consistent pagination, entity and timeframe selection across all v2 API endpoints
  - Full control over the resulting payload and information
  - Common UI backlink

Query the types of entities available

- *https://{DynatraceURL}/api/v2/entityTypes*
- *https://{DynatraceURL}/api/v2/entityTypes/{type}*

- The resulting JSON payload, as shown below, returns 22 individual entity types along with all their properties.
- If we focus on a specific entity type, such as hosts we can see all the relevant properties, tags, and management zone information that Dynatrace collects for the hosts and the relationships that these hosts have with other parts of the topology



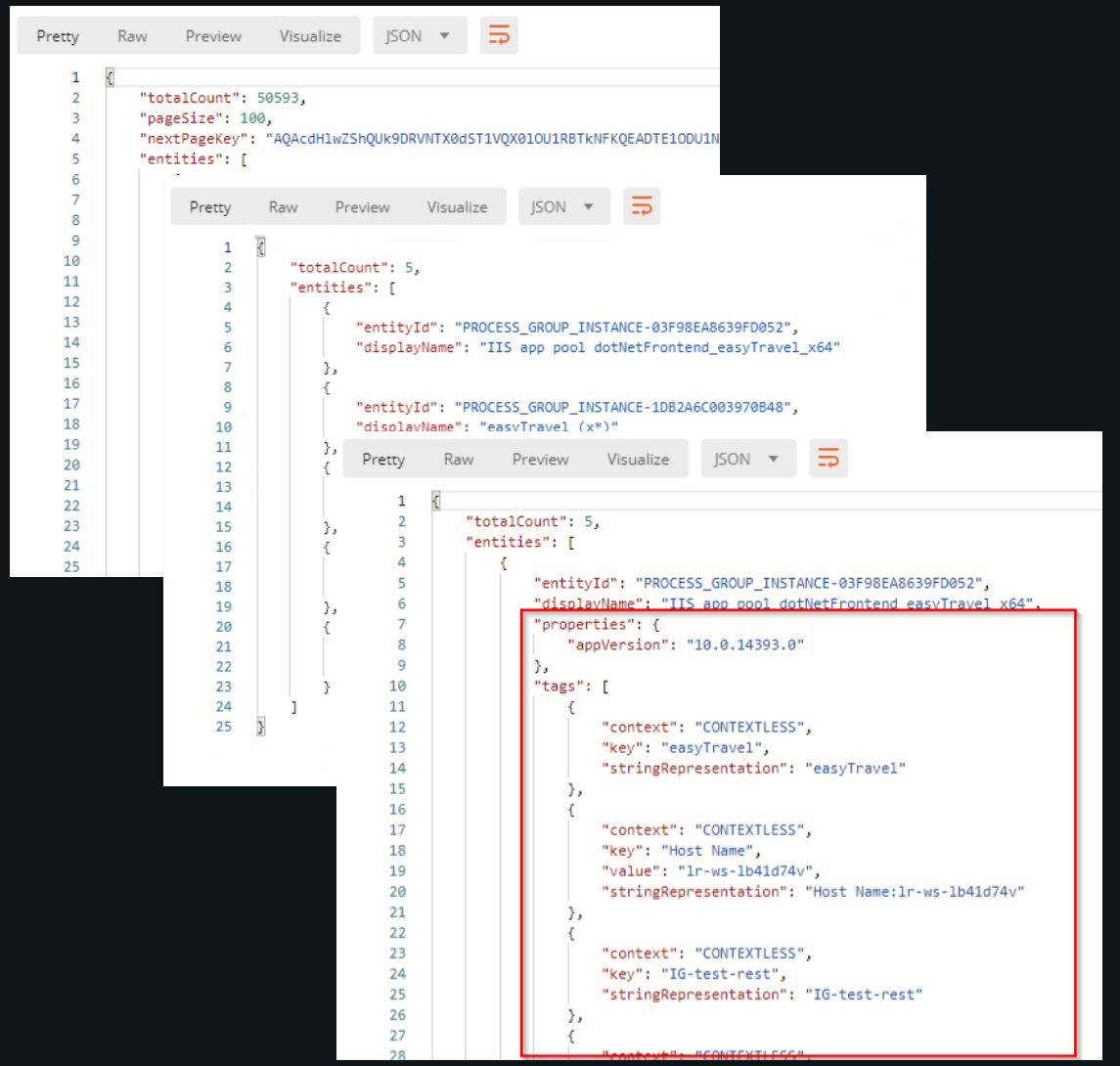


Query a filtered list of entities

- *https://{DynatraceURL}/api/v2/entityTypes/{type}*

- A call to this endpoint will return a list of all entities of the specified type, and can accept multiple {type} filters

- A simple query with *entitySelector=type(PROCESS_GROUP_INSTANCE)* we see all PGIs in the environment. Default page size is 100.
- We can filter this down by passing additional filters: entitySelector=type(PROCESS_GROUP_INSTANCE),mz(Easytravel), tag(loadtest),entityName(easy)
- We can also increase the amount of data we get back for each entity by adding the *fields* parameter: fields=+tags,+properties.appVersion



Entity API v2

- See complete Monitored entities API v2 here:
- https://www.dynatrace.com/support/help/dynatrace-api/environment-api/entity-v2/

Events API

Events API

- *https://{DynatraceURL}/api/v1/events/*

- This family of endpoints delivers details about all uncorrelated events that Dynatrace collects within a given environment
- Information returned for each event includes attributes about the event source, the entity where the event was collected, and other event-specific details
- This family of endpoints allows API consumers to receive a global feed of all uncorrelated events in an environment and to push external events to a Dynatrace environment

Read the Events feed

- This endpoint is limited to a maximum of 500 single events per request
- This means that API consumers must use filters to further focus their queries, either on specific monitored entities or to requests that occurred during a specific time frame
- The following request parameters can be used to filter requests to the events endpoint:
  - **from, to**: Specifies a time frame for the query (default is the last hour)
  - **eventType**: Filter the event feed based on a specific event type (for example, UNEXPECTED_LOW_LOAD)
  - **entityId**: Only receive events for a given monitored entity, such as a host, process, or service

Result

- Query results are returned as JSON payloads
- Each event contains at least the following meta information:
  - **eventId**: Unique environment identifier for a specific event.
  - **startTime**, **endTime**: Timestamp when detected/closed
  - **entityId**, **entityName**: Unique identifier/name of the monitored entity
  - **severityLevel**: Denotes the severity level of an event
  - **impactLevel**: Denotes the impact level of the event
  - **eventType**: Specifies the event type
  - **eventStatus**: Specifies the event state (either OPEN or CLOSED)
  - **tags**: Collects all the tags associated with the monitored entity
  - **source**: Defines the source of the raised event


```
{
  events: [
    {
      eventId: -5106087015642687000,
      startTime: 1496246212169,
      endTime: 1496299458208,
      entityId: "HOST-81C88DBE606D52A6",
      entityName: "Host 123",
      severityLevel: "AVAILABILITY",
      impactLevel: "INFRASTRUCTURE",
      eventType: "CONNECTION_LOST",
      eventStatus: "CLOSED",
      tags: null,
      source: "builtin"
    },
```

Generate events and trigger alerts with third-party tools

- Third-party API clients and tools can now create all types of severity events and thereby trigger the creation and correlation of alerts for new problems.
- Example
  - Imagine you receive an alert from your UPS notifying you that one of the server rooms in your data center has suffered a power outage. Or in the unusual case that you need to use a legacy network monitoring system to alert you of network problems. You may want to push these events into Dynatrace to take advantage of the intelligent problem correlation Dynatrace Provides.
- It's important to note that all external events must target at least one Smartscape component, which can be a host or other component, such as a service, application, or even a single process.

User Session Query Language (USQL)

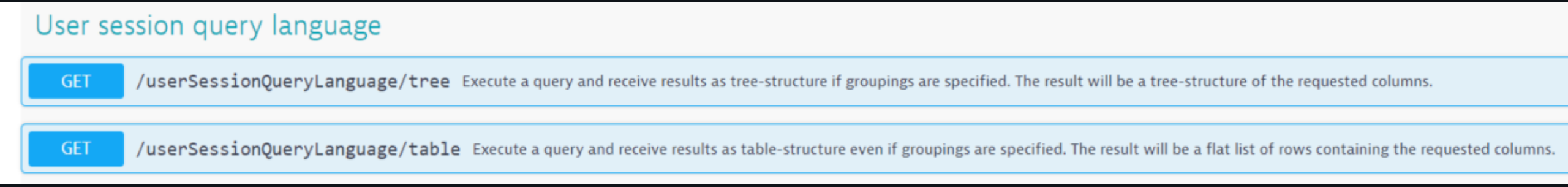
User Session Query Language

- *https://{DynatraceURL}/api/v1/userSessionQueryLanguage/*

- Dynatrace captures user session data, which includes customer behavior and high level performance data
- The User Session Query Language (USQL) provides an easy way to query analytics and performance data captured by Dynatrace and run powerful queries, segmentations, and aggregations on top of this user session data
- The USQL is not SQL, as Dynatrace does not store the data in a relational database
- The USQL does use many of the concepts of SQL and the syntax is similar in order to make it easy to get started with, but it's a Dynatrace specific query language

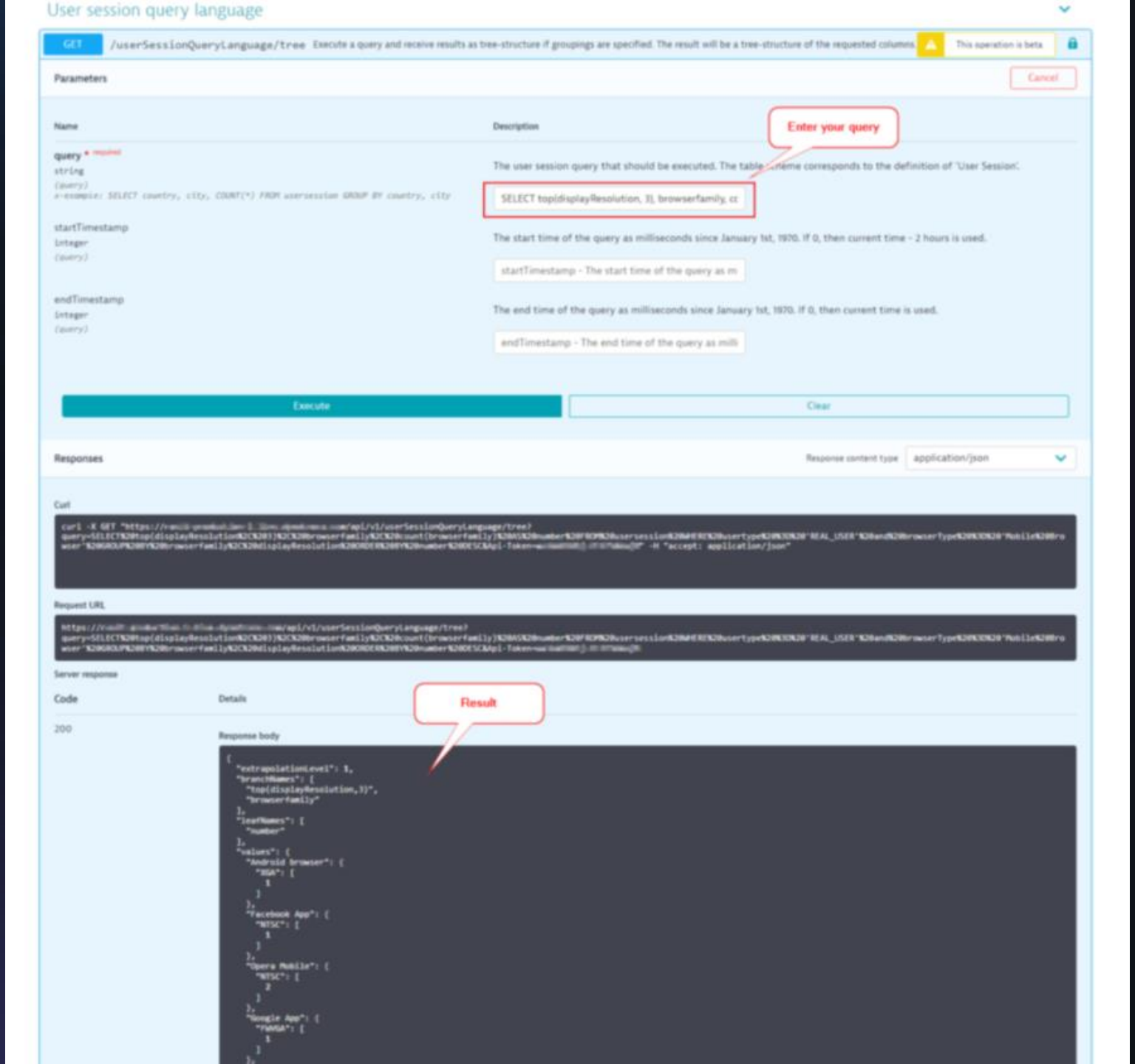
User Session Query Language

- The USQL can be accessed by using the Dynatrace API Explorer



User Session Query Language

- For this example, let's query the top 3 mobile-device display resolutions used by real users
- The results should be organized by browser family and ordered, as shown in the following query:
  - *SELECT top(displayResolution, 3), browserfamily, count(browserfamily) AS number FROM usersession WHERE usertype = 'REAL_USER' and browserType = 'Mobile Browser' GROUP BY browserfamily, displayResolution ORDER BY number DESC*


User session query language
GET
/userSessionQueryLanguage/tree
Execute a query and receive results as tree-structure if groupings are specified. The result will be a tree-structure of the requested columns.
This operation is beta
Parameters
Cancel
Name
Description
Enter your query
query * required
string
(query)
e.example: SELECT country, city, COUNT(*) FROM usersession GROUP BY country, city
The user session query that should be executed. The table schema corresponds to the definition of 'User Session'.
SELECT top(displayResolution, 3), browserfamily, cc
startTimestamp
integer
(query)
The start time of the query as milliseconds since January 1st, 1970. If 0, then current time - 2 hours is used.
startTimestamp - The start time of the query as m
endTimestamp
integer
(query)
The end time of the query as milliseconds since January 1st, 1970. If 0, then current time is used.
endTimestamp - The end time of the query as milli
Execute
Clear
Responses
Response content type
application/json
Curl
Request URL
Server response
Code
Details
Result
200
Response body
{
"extrapolationLevel": 1,
"branchNames": [
"top(displayResolution,3)",
"browserfamily"
],
"leafNames": [
"number"
],
"values": {
"Android browser": {
"USA": [
1
]
},
"Facebook App": {
"NTSC": [
1
]
},
"Opera Mobile": {
"NTSC": [
2
]
},
"Google App": {
"PANDA": [
1
]

User Session Query Language

- USQL offers you two tables: User sessions and User actions
- You can find all the details in Dynatrace Help and by looking at the models in API explorer view
- Here two more examples:
  - Example 1: Give me the average user action duration, visually complete, and DOM complete time for all frustrated load user actions that are not part of a bounced session
  - Note that usersession is used to access user session attributes when selecting data from user actions
    - *SELECT avg(duration), avg(visuallyCompleteTime), avg(domCompleteTime) FROM useraction WHERE usersession.bounce is false AND apdexCategory = 'FRUSTRATED' AND type = 'Load'*
  - Example 2: I want to know the number of named users of mobile applications who have accessed a user action with the name "Touch on Search"
    - *SELECT count(userid) FROM usersession WHERE applicationType = 'MOBILE_APPLICATION' and useraction.name = 'Touch on Search'*

Log Analytics

Log Analytics

- The Log Analytics API is only available for Dynatrace environments that have a paid Log Analytics license
- Using the Log Analytics API you can:
  - Get log lists based on host ID or process group
  - Start a log retrieval job based on host ID or process group
    - Optionally, you can indicate a start and end timestamp or query to filter log content
  - Get the status of a log retrieval job based on host ID or process group
  - Get the content of a log based on host ID or process group
  - Delete or cancel a log retrieval job based on host ID or process group

Log Analytics Examples

- Get list of logs for a specific host
  - curl -i -k -H "Authorization: Api-Token eQBpoCtIRCSaAyjrethyf" https://myhost:8021/e/1/api/v1/entity/infrastructure/hosts/HOST-EA474F61FFBD5C97/logs
- Start a log retrieval job
  - curl -X POST -i -k -H "Authorization: Api-Token eQBpoCtIRCSaAyjrethyf" https://localhost:8021/e/1/api/v1/entity/infrastructure/hosts/HOST-EA474F61FFBD5C97/logs/%2Fvar%2Flog%2Fsyslog?query=info
- Get the content of a log
  - curl -i -k -H "Authorization: Api-Token eQBpoCtIRCSaAyjrethyf" https://localhost:8021/e/1/api/v1/entity/infrastructure/hosts/HOST-EA474F61FFBD5C97/logs/jobs/707306f2-f3c2-4f7b-a457-cf00f7a65b1d/records?pageSize=1&scrollToken=63

RUM JS API

RUM JS API

- *https://{DynatraceURL}/api/v1/rum/*

- The RUM JavaScript management API provides endpoints to help you set up and maintain your manually injected applications
- You can fetch detailed information about your existing manually injected applications by calling the *manualApps* endpoint
- The *jsLatestVersion* and *appRevision* endpoints deliver version information that indicates if the JavaScript code used in each of your manually injected applications is still up-to-date
- Finally, the *jsTag* and *jsInlineScript* endpoints return the JavaScript code that you must insert into each of your applications' pages to enable real user monitoring.

Manually injected applications information Endpoint

- *https://{DynatraceURL}/api/v1/rum/manualApps*

- A call to the /rum/manualApps endpoint returns a list that contains detailed information about each of your manually injected applications.
- Result
  - **applicationId**: The unique application ID generated by Dynatrace.
  - **displayName**: The name of the application assigned by the user.
  - **monitoringEnabled**: True if user monitoring is enabled, False if it isn't monitored.
  - **revision**: Timestamp of the last application update.
  - **deleted**: True if the application has been deleted, False if it hasn't been deleted.

Latest JavaScript code version Endpoint

- *https://{DynatraceURL}/api/v1/rum/jsLatestVersion*

- A call to the /rum/jsLatestVersion endpoint returns the current real user monitoring JavaScript code version

Current Application Revision Endpoint

- *https://{DynatraceURL}/api/v1/rum/appRevision/{applicationid}*

- A call to the /rum/appRevision/{applicationId} endpoint returns the current application's real user monitoring version

JavaScript code Endpoint

- *https://{DynatraceURL}/api/v1/rum/jsTag/{applicationid}*
- *https://{DynatraceURL}/api/v1/rum/jsInlineScript/{applicationid}*

- You can use one of the two endpoints to fetch the JavaScript code from one of your manually injected applications

Enhance Dynatrace Data

Custom devices and custom metrics API

Custom devices and custom metrics API

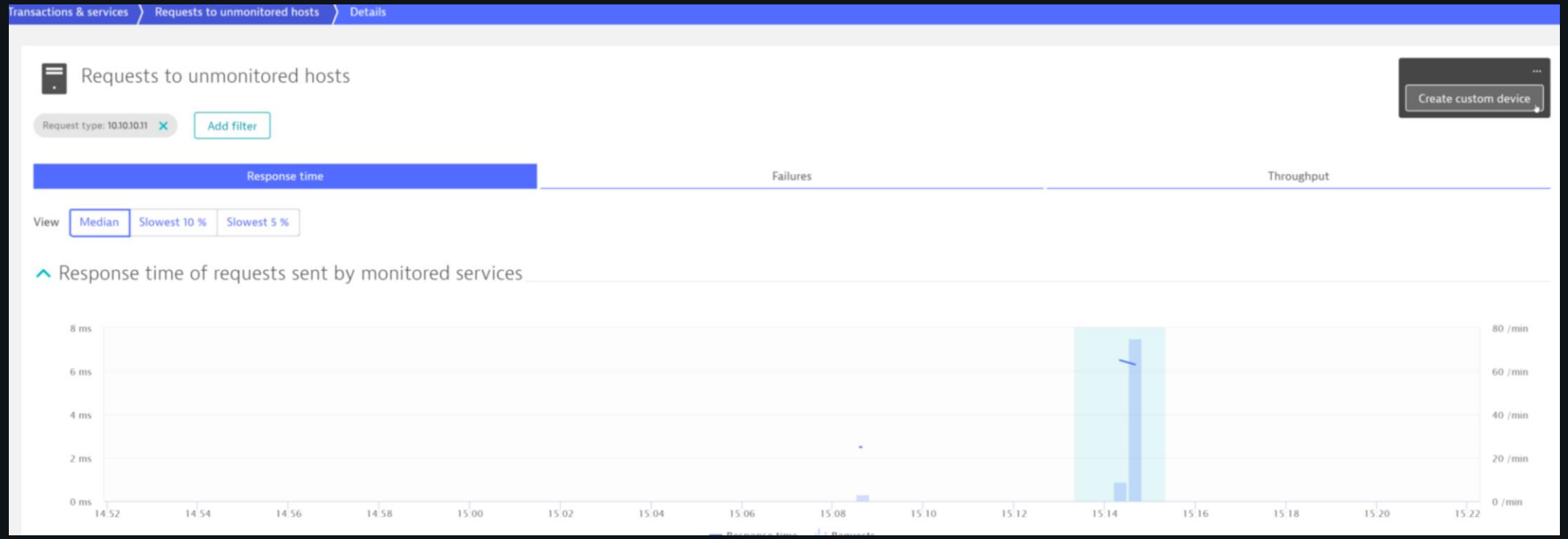
- The custom network devices and metrics API allows users to create new kinds of network components and to register and send custom metrics for these devices
  - A custom network device is any part of your environment that can't run the OneAgent
  - Examples of such network devices include Firewalls, DataPower gateways, cloud databases, and any other network appliance such as a proxy or gateway
- By using this API, it's possible for your own networked box to send custom metrics into Dynatrace based on the native properties of these devices, or to write your own scripts that send the metrics in place of the network or cloud network box
- Dynatrace is automatically able to map your custom device into your existing Smartscape environment

Create a custom device from existing requests

- You may find a service labeled 'Requests to unmonitored hosts' in your environment. This indicates outbound requests from monitored service are communicating with another service / device that is not monitored by Dynatrace OneAgent.
- In cases where you cannot install a OneAgent on a device or host, you can now define one automatically!
- This feature also allows requests within 'Requests to Unmonitored hosts' be tracked independently as their own service.

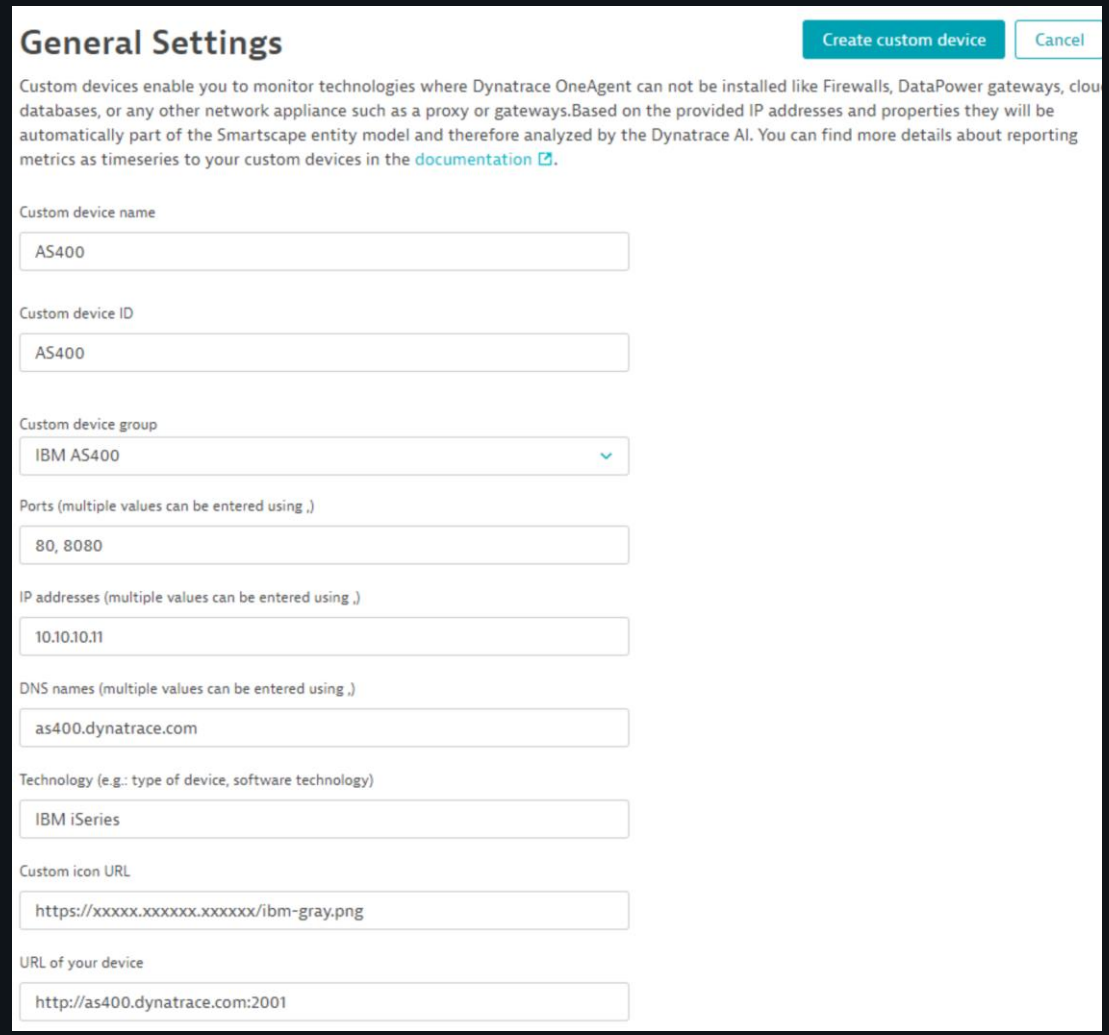
Create custom device from existing requests

- Navigate to a specific request within the 'Requests to Unmonitored hosts' service
- With the filter applied a new option is now available in the top right to define a custom device



Custom Device settings

- The Custom device ID needs to be the **unique** ID of that custom device and will also be part of the API endpoint to report metrics to Dynatrace – this field is already prefilled with a generic ID , but you may change it to something more readable (but still unique in your environment).
- Each device need to be part of a device group
- Define a technology
- You can use custom icon images for UI display



List all custom metrics

- *https://{DynatraceURL}/api/v1/timeseries?filter=CUSTOM*

- The timeseries endpoint covers all metric-relevant information. A HTTP GET request lists all available metric definitions within your given environment.
- Use the filter parameter (BUILTIN, PLUGIN, or CUSTOM) to list only the custom-defined metrics
- See the previous 'timeseries' API section


```
[
  {
    timeseriesId: "custom:firewall.connections.dropped",
    displayName: "Dropped TCP connections",
    dimensions: [
      "CUSTOM_DEVICE"
    ],
    aggregationTypes: [
      "AVG",
      "SUM",
      "MIN",
      "MAX"
    ],
    unit: "%",
    filter: "CUSTOM",
    types: [
      "F5-Firewall"
    ]
  }
]
```

Register your custom metric

- *https://{DynatraceURL}/api/v1/timeseries/custom:name.of.metric*

- The same timeseries endpoint is used to register new custom metrics by sending a HTTP PUT call, where the payload contains the metadata for your new metric
- Metrics must be assigned a software technology type that is identical to the technology type of the custom device you are sending the metric to
- Result
  - displayName: The display name of the metric that will be shown in the UI
  - unit: The definition of the unit that will be used for this metric
  - dimensions: Definition of a metric dimension key that is used to report multiple dimensions
  - types: List of technology types for which this metric is registered

```
"displayName" : "Dropped TCP connections",
"unit" : "Count",
"dimensions": [
        "nic"
],
"types": [
        "F5-Firewall"
]
```

Delete a custom metric

- *https://{DynatraceURL}/api/v1/timeseries/custom:name.of.metric*

- You can delete your custom metric using a HTTP DELETE request on a custom metric id
- If you delete a metric definition, you will lose any metrics that you sent in earlier

Start reporting metrics for a custom device

- *https://{DynatraceURL}/api/v1/entity/infrastructure/custom/unique.identifier*

- After you've registered all your custom metrics, it's time to send metric values to a new custom device
- You don't need to register a custom device
- Just start sending information and the device will be created automatically
- To send metric information for a custom device, use the topology endpoint and push custom device meta information as well as all metric information through a HTTP POST request

Start reporting metrics for a custom device

- Payload
  - displayName: The display name of the component that will be used within the UI
  - ipAddresses: List of IP addresses that belong to the component
  - listenPorts: More detailed information about how this component accepts communication from other components on the network-port level
  - type: A defined software technology type for this custom component
  - favIcon: A URL where we can fetch a grayscale or monochrome icon that will be used for your custom component
  - configURL: A URL used to directly link to the configuration web page of the entity
  - tags: List of custom user labels that you want attached to your custom component
  - properties: A list of key value pair properties that will be shown beneath the infographics of your custom component
  - series: List of metric values that are reported for the custom component

Example

```
{
  "displayName" : "F5 Firewall 24",
  "ipAddresses" : ["172.16.115.211"],
  "listenPorts" : ["9999"],
  "type" : "F5-Firewall",
  "favicon" : "http://assets.dynatrace.com/global/icons/f5.png",
  "configUrl" : "http://172.16.115.211:8080",
  "tags": ["user defined tag", "tag2"],
  "properties" : { "prop1" : "propvalue" },
  "series" : [
    {
      "timeseriesId" : "custom:firewall.connections.dropped",
      "dimensions" : { "nic" : "ethernetcard1"  },
      "dataPoints" : [ [ <timestamp as a number, for example: 1495520570871>  , <value> ] ]
    },
    {
      "timeseriesId" : "custom:firewall.connections.dropped",
      "dimensions" : { "nic" : "ethernetcard2"  },
      "dataPoints" : [ [ <timestamp as a number, for example: 1495520570871>  , <value> ] ]
    }
  ]
}
```

Events API

Events API

- *https://{DynatraceURL}/api/v1/events/*

- This family of endpoints delivers details about all uncorrelated events that Dynatrace collects within a given environment
- Information returned for each event includes attributes about the event source, the entity where the event was collected, and other event-specific details
- This family of endpoints allows API consumers to receive a global feed of all uncorrelated events in an environment and to push external events to a Dynatrace environment
- The events REST endpoint enables 3rd party integrations to push custom events to one or more monitored entities via the API

Push custom events from 3rd party systems

- The events REST endpoint enables 3rd party integrations to push custom events to one or more monitored entities via the API
- The intent of this interface is to allow 3rd party systems, such as CI platforms (Bamboo, Electric Cloud, etc) to provide additional detail for Dynatrace automated root cause analysis
- The events API offers a set of semantically predefined event types that allow the Dynatrace problem correlation engine to correctly handle information provided by external systems

Push custom events from 3rd party systems

- Each event, regardless of type, must provide the following common meta information:
  - **eventType**: The predefined event type, such as 'CUSTOM_DEPLOYMENT' or 'CUSTOM_ANNOTATION'
  - **start**: Optional start timestamp of the event in UTC milliseconds
    - The actual time is used for the event if no start timestamp is provided
  - **end**: Optional end timestamp of the event in UTC milliseconds
    - The actual time is used if no end timestamp is provided
  - **attachRules**: A complex structure that contains attachment rules that define which monitored entities the event is to be attached to

```
"attachRules": {
  "entityIds" : ["APPLICATION-B66B773D12C49189"],
    "tagRule" : [{
        "meTypes" : ["SERVICE"],
        "tags" : ["PRODUCTION"]
    }]
```

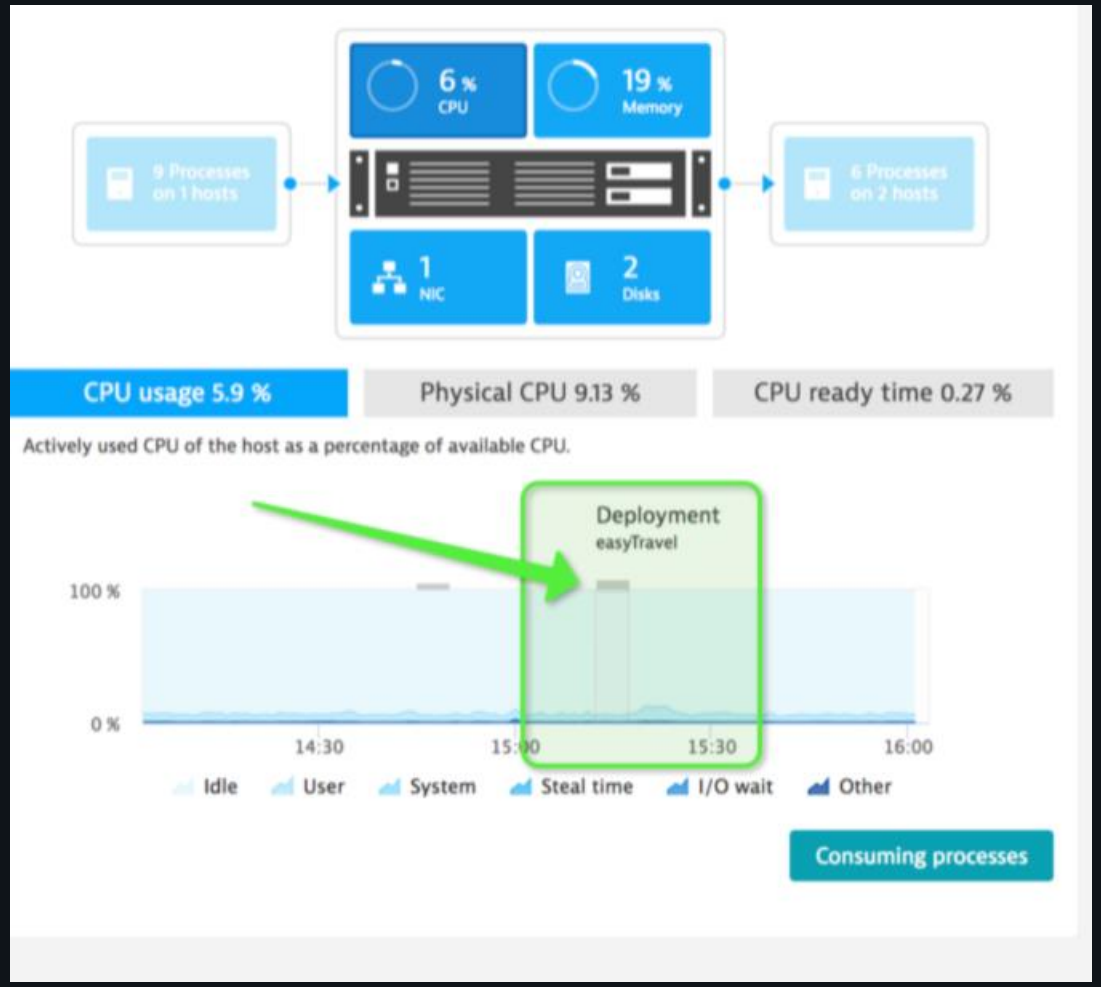
Push custom events from 3rd party systems

- Deployment Events ('CUSTOM_DEPLOYMENT')
- This event type is used to inform a Dynatrace environment of a deployment that is triggered by continuous integration or IT automation tools
- Deployment events can provide the following predefined meta information:
  - **deploymentName**: A string identifier for the deployment that was triggered
  - **deploymentVersion**: A version string for the deployment
  - **deploymentProject**: Project name of the deployed package
  - **remediationAction**: Link to the deployment related remediation action within the external tool
  - **ciBackLink**: Link to the deployed artifact within the 3rd party system
  - **source**: Specifies the name or identifier for the external tool that provided the custom deployment event
  - **timeseriesIds**: Optional array of timeseries IDs that are related to the event
  - **customProperties**: Dictionary of custom key value pairs that can be used to send additional information

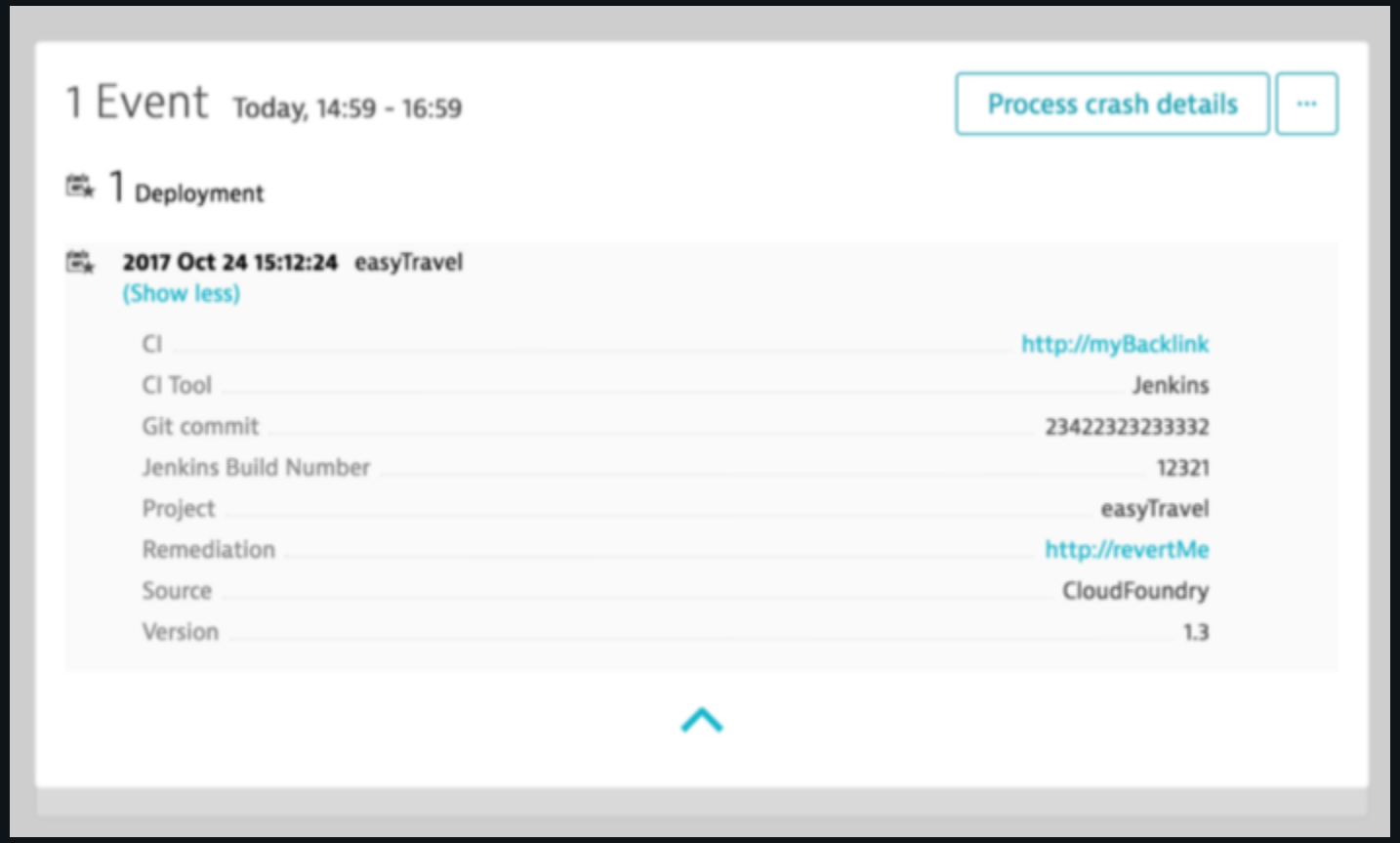
Example

```
{
  "start":1495200637630,
  "end":1495200637630,
  "eventType": "CUSTOM_DEPLOYMENT",
  "attachRules": {
    "entityIds" : ["APPLICATION-B66B773D12C49189"],
      "tagRule" : [{
        "meTypes" : ["SERVICE"],
        "tags" : ["PRODUCTION"]
      }]
  },
  "deploymentName":"easyTravel",
  "deploymentVersion":"1.0",
  "deploymentProject":"easyTravel",
  "remediationAction":"http://revertMe",
  "ciBackLink":"http://myBacklink",
  "source":"CloudFoundry",
  "customProperties":{
    "CI Tool": "Jenkins",
    "Jenkins Build Number": "12321",
    "Git commit": "23422323233332"
  }
}
```

Push custom events from 3rd party systems



Push custom events from 3rd party systems



Push custom events from 3rd party systems

- Annotation Events ('CUSTOM_ANNOTATION')
- This event type is used to annotate a monitored entity within Dynatrace
- Possible use cases include informing about important changes that were made within the monitored environment
- Annotations can provide the following predefined meta information:
  - **annotationType**: A string identifier for the type that is used to group the annotations
  - **annotationDescription**: A textual description of the annotation
  - **source**: Specifies the name or identifier of the external tool that provided the custom annotation event
  - **customProperties**: Dictionary of custom key value pairs that can be used to send additional information

Example

- This example shows how to push a new annotation event to one of your applications

```
{
  "start":1495200637630,
  "end":1495200637630,
  "eventType": "CUSTOM_ANNOTATION",
  "attachRules":
  {
    "entityIds": [ "APPLICATION-B66B773D12C49189" ]
  },
  "annotationType" : "DNS route changed",
  "source" : "OpsControl",
  "annotationDescription" : "We changed the DNS configuration",
  "timeseriesIds":[],
  "customProperties":
  {
    "original": "x.rxlab.com",
    "changed": "x.dtlab.com"
  }
}
```

Tags

Tagging Endpoint

- *https://{DynatraceURL}/api/v1/entity/services/{YOUR_SERVICE_ID}*

- The Smartscape and topology API not only offers REST endpoints for fetching information related to all the monitored components in your environment, it also allows you to assign tags to individual components
- By using the same REST endpoints as described above, it's possible to perform an HTTP POST call to push a list of tags to a given entity ID, as shown in the example below.

```
{
  "tags": ["myTagthroughAPI", "secondTag"]
}
```

Configuration

Maintenance Windows

Maintenance Windows

- *https://{DynatraceURL}/api/v1/maintenance/*

- The different endpoints within the Configuration API allow you to read, create, or modify settings within a Dynatrace environment
- The maintenance window endpoint allows API consumers to automate the creation, update, and deletion of maintenance windows within a monitored environment
- The thresholds REST endpoint enables 3rd party tools to manage all configured plugin and custom event thresholds within a monitored environment

View all configured maintenance windows

- *https://{DynatraceURL}/api/v1/maintenance/{mwid}*

- Return the details of a specific maintenance windows
- Remove {mwid} to return the details of all maintenance windows

- **from, to**: Specifies a time frame for the query as a Unix Epoch timestamp in milliseconds
- **type**: Values are Planned or Unplanned.


```
id: "New application deployment",
type: "Planned",
description: "We will deploy a new easyTravel application version",
suppressAlerts: false,
suppressProblems: false,
scope: null,
schedule: {
  type: "Day",
  timezoneId: "Europe/Vienna",
  maintenanceStart: "2017-08-29 14:43",
  maintenanceEnd: "2017-08-29 15:43",
  recurrence: {
    start: "14:43",
    duration: 556
  }
```

View all configured maintenance windows

- Result
  - id: Unique maintenance window identifier
  - type: The type of the downtime either Planned or Unplanned
  - description: Textual description of a maintenance window
  - suppressAlerts: Suppresses the sending of alerts for problems detected during maintenance windows
  - suppressProblems: Disables problem detection within the scope of the maintenance window
  - scope: The defined scope of the maintenance window
  - schedule: Specifies the schedule of the maintenance window
    - Can be either a one-time window or a recurring schedule of maintenance windows

Create a maintenance window

- *https://{DynatraceURL}/api/v1/maintenance/*

- API consumers can create or update maintenance windows for periods of planned system downtime
- Each maintenance window can have its own unique identifier

- Use an HTTP POST request on the maintenance API and include values for each of the properties

```
"id" : "my.unique.mwindow.id.1",
"type": "Planned",
"description" : "Here comes a detailed description of the maintenance window",
"suppressAlerts" : true,
"suppressProblems" : false,
"scope" : {
        "entities" : [
                "HOST-0B3371A5AC53FF12", "SERVICE-13FA1F30530CDEE1"
        ],
        "matches" : [
                {
                        "type" : "HOST",
                        "tags" : [
                                {
                                        "context" : "AWS",
                                        "key" : "myTag1",
                                        "value" : "myValue1"
                                },
                                {       "key" : "myTag2" }
                        ]
```

Delete a maintenance window

- *https://{DynatraceURL}/api/v1/maintenance/{mwid}*

- API consumers can delete maintenance windows
- Each maintenance window can have its own unique identifier

- Use an HTTP DELETE request on the maintenance API and include the given identifier

Plugins and Custom Events

Read all plugin and custom events thresholds

- *https://{DynatraceURL}/api/v1/thresholds/*

- API consumers can read all configured plugin and custom event thresholds within a monitored environment

- Use an HTTP GET request to return the details of all plugin and custom event thresholds

Read all plugin and custom events thresholds

- Result
  - **thresholdId**: Unique identifier that is used to update or delete the threshold
  - **timeseriesId**: Unique identifier of the metric that the threshold applies to
  - **threshold**: The actual threshold that should be checked
  - **alertCondition**: Alert if actual metric value is either ABOVE or BELOW the defined threshold
  - **samples**: Number of minute samples that defines the sliding window
  - **violatingSamples**: Number of minute sliding window samples that have to violate within the sliding window
  - **dealertingSamples**: Number of minute samples that have to return to normal before the event is closed again
  - **eventType**: Semantic event type
  - **eventName**: The human readable name of the event
  - **filter**: Shows the source of the threshold that can either be API or a PLUGIN
  - **description**: A placeholder filled human readable description of the event

Example

```
[
  {
    thresholdId: "newthreshold",
    timeseriesId: "ruxit.jmx.Hadoop.yarn:AllocatedMB",
    threshold: 4,
    alertCondition: "ABOVE",
    samples: 5,
    violatingSamples: 2,
    dealertingSamples: 3,
    eventType: "ERROR_EVENT",
    eventName: "High allocated memory",
    filter: "API",
    description: "My own description of the event"
  },
  {
    thresholdId: "ruxit.python.haproxy:be_susage:be_usage_alert_high_generic",
    timeseriesId: "ruxit.python.haproxy:be_susage",
    threshold: 80,
    alertCondition: "ABOVE",
    samples: 5,
    violatingSamples: 3,
    dealertingSamples: 5,
    eventType: "PERFORMANCE_EVENT",
    eventName: "High HAProxy backend session usage",
    filter: "PLUGIN",
    description: "The backend session usage is {alert_condition} the threshold of {threshold}"
  }
]
```

Create/update a plugin and custom event threshold

- *https://{DynatraceURL}/api/v1/thresholds/{identifier}*

- API consumers can manage all configured plugin and custom event thresholds within a monitored environment

- Use an HTTP PUT request to create or update a custom event threshold with the identifier and include values for each of the properties


```
"timeseriesId": "custom:raspberry.files.count",
"threshold": 3,
"alertCondition": "ABOVE",
"samples": 1,
"violatingSamples": 1,
"dealertingSamples": 1,
"eventType": "ERROR_EVENT",
"eventName": "Motion detected",
"description": "Oh we dicovered some motion of {severity}
```

Delete a plugin and custom event threshold

- *https://{DynatraceURL}/api/v1/thresholds/{identifier}*

- API consumers can delete all configured plugin and custom event thresholds within a monitored environment

- Use an HTTP DELETE request to delete a selected threshold

Read and Write Configuration

Read and Write Configuration

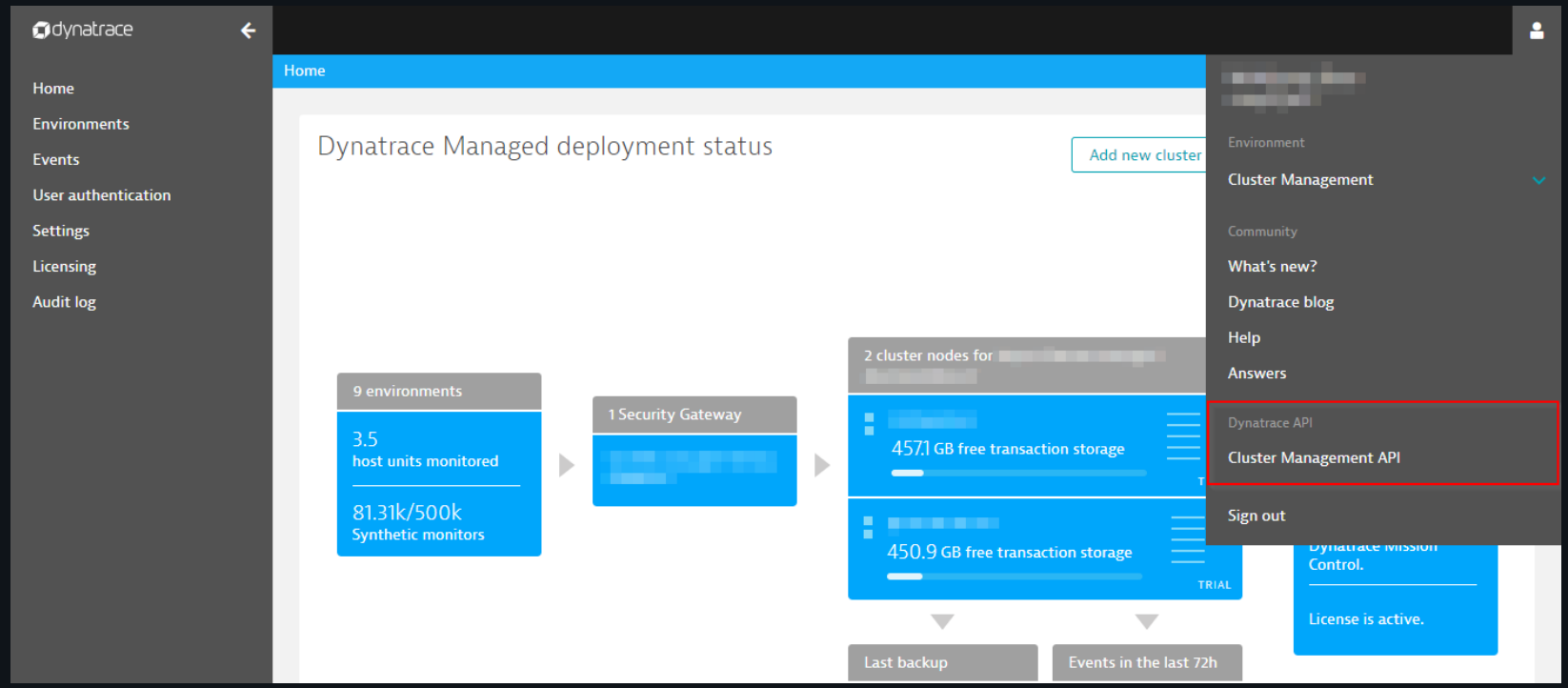
- *https://{DynatraceURL}/api/config/v1/*

- The Dynatrace configuration API is intended to allow API consumers to read and write the global configuration of an existing Dynatrace environment
- Some API endpoints are:
  - Alerting profiles
  - Anomaly Detection
  - Automatically applied tags
  - Cloud Integration Credentials
  - Calculated metrics
  - OneAgent Configurations
  - RUM Configurations
  - Services Configurations

Cluster Management API

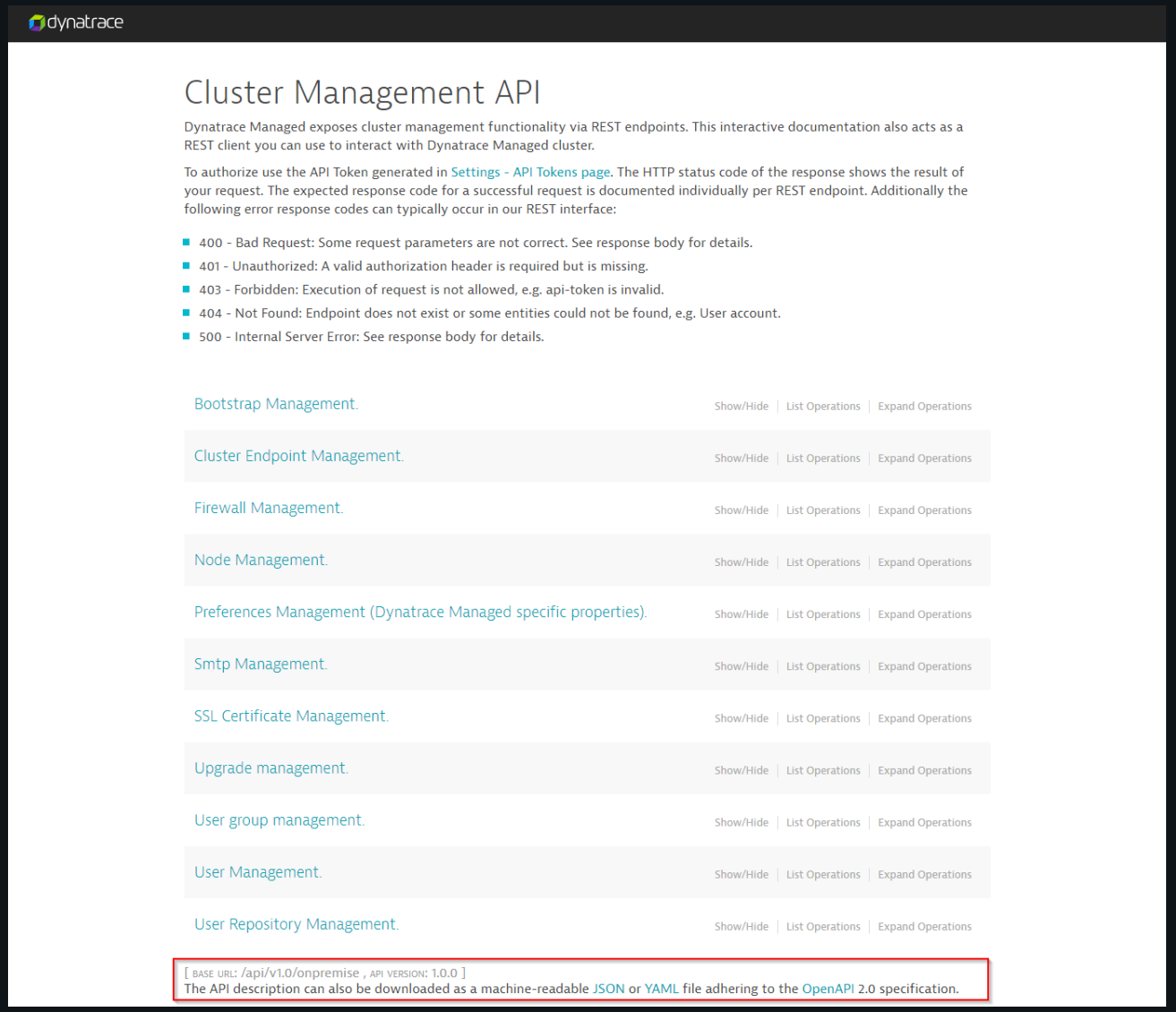
Access Cluster Management API

- A click into the cluster management API menu directly jumps into the interactive API documentation.



Documentation

- The documentation lists all available REST endpoints with their methods along with possible input and output data type definitions.
- All changes pushed through the API are real and will have an immediate effect on your running Dynatrace Managed cluster.

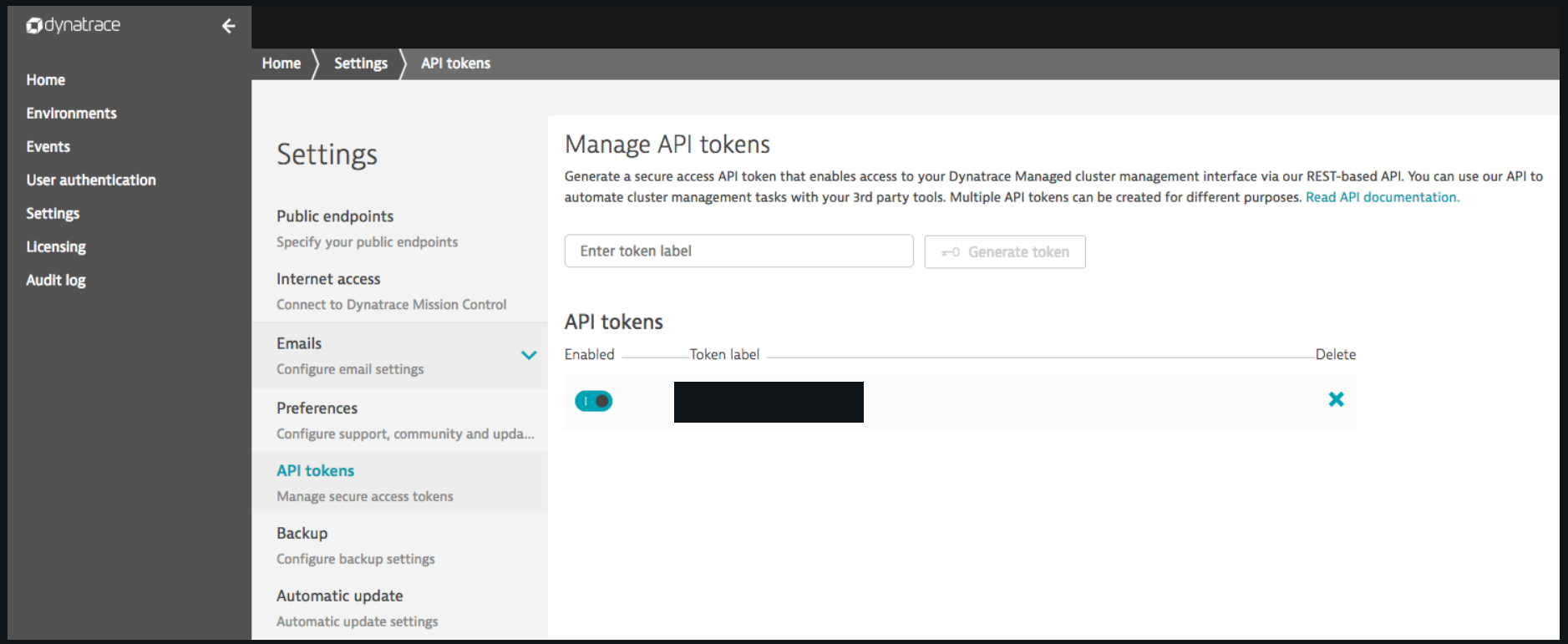


Use of the Managed API

- Environment management
- Configuration management
- Cluster Management
- User/Group management

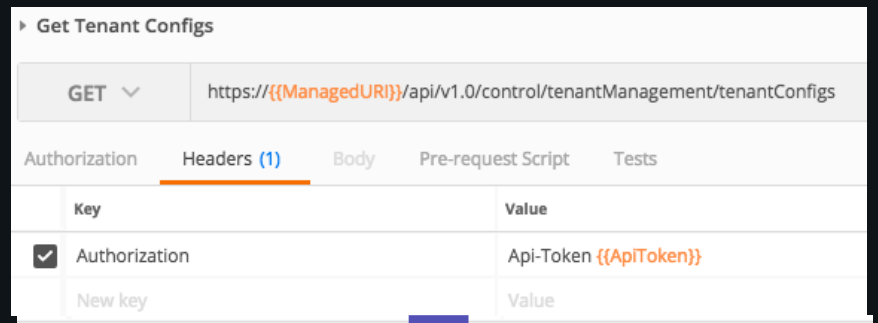
Requirements for the Managed API

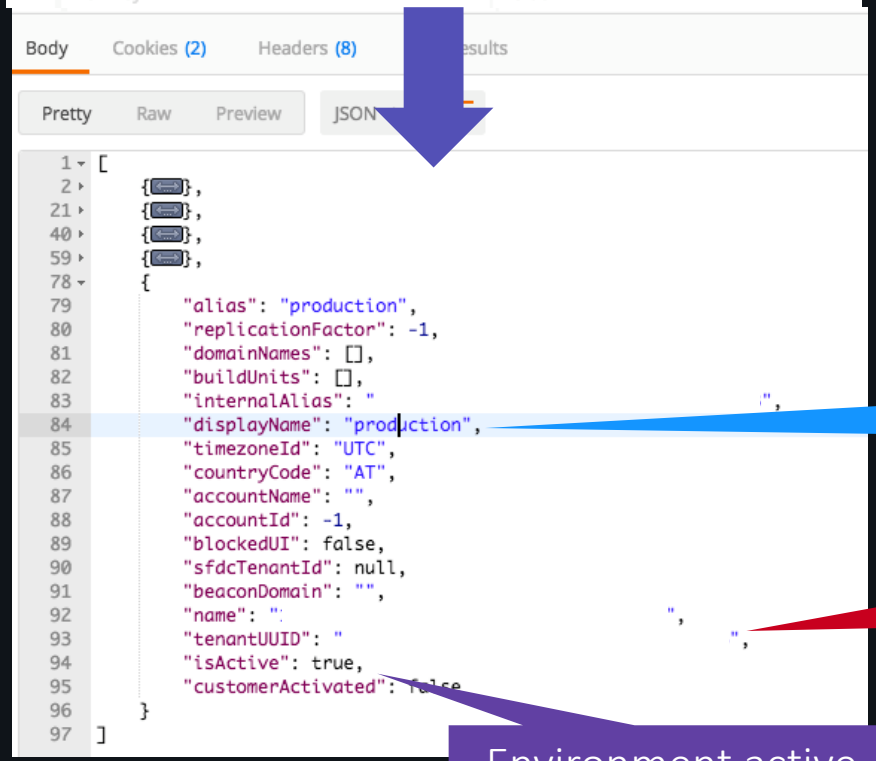
- Cluster API token ➔ NOT the same as an environment API token!

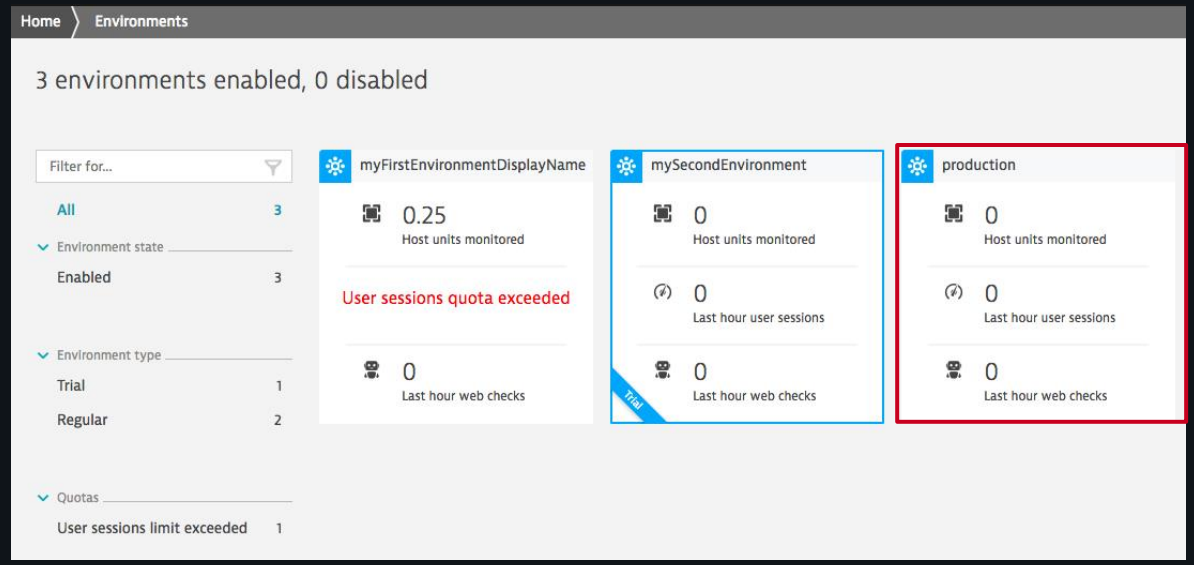


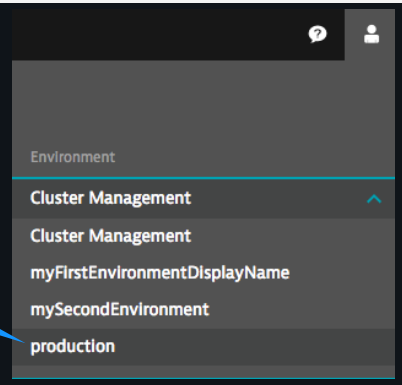
Get environment information

To see which environments are on a cluster









Name visible inside environment selection

Environment ID visible in browser

Environment active or not

https:// {{ManagedURI}} /e/ {{Tenant UUID}} /#dashboard

Getting Environment Quota Information

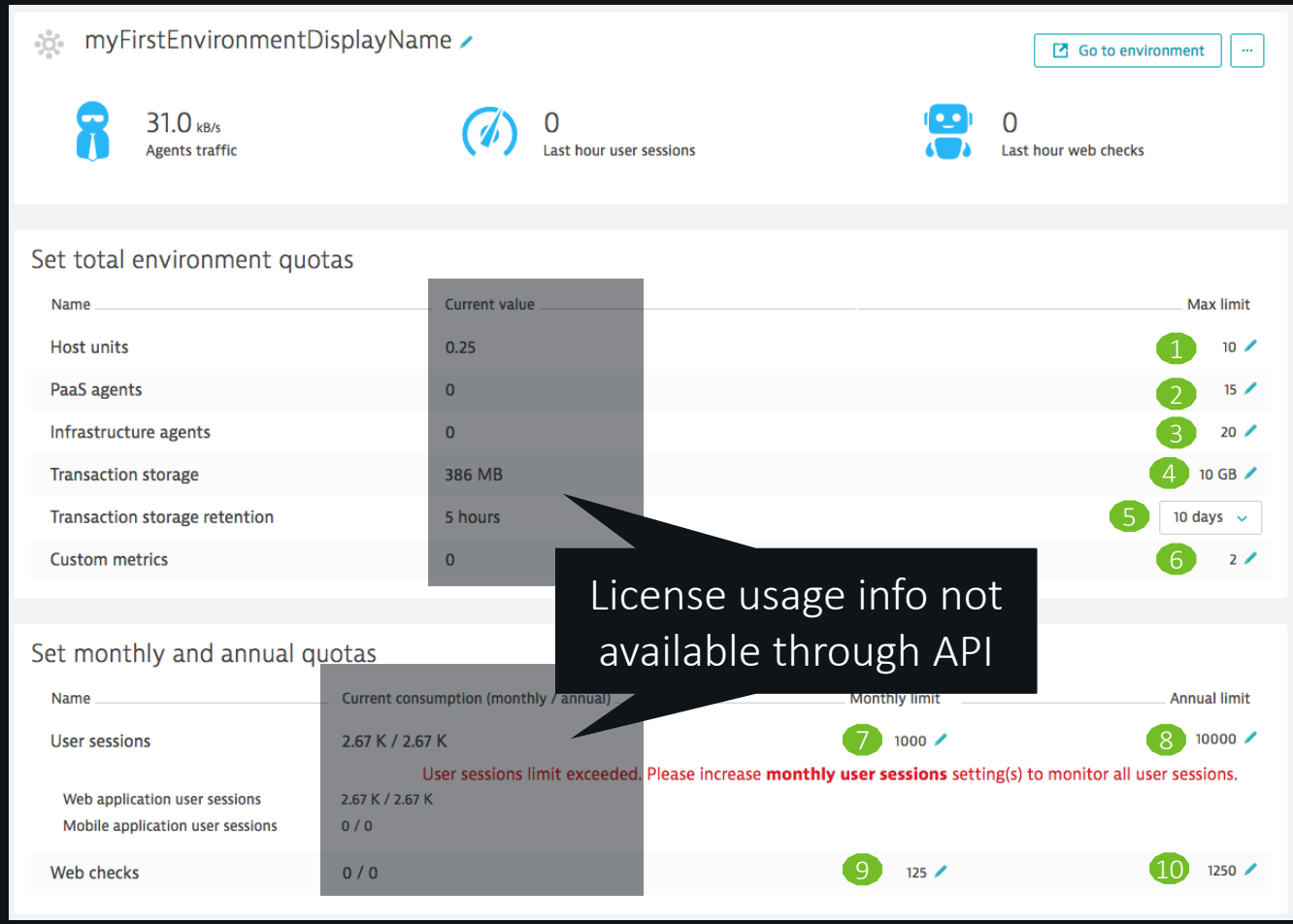
Get License - 1 Tenant

GET https://{{ManagedURI}}/api/v1.0/control/tenantManagement/license/{{Environment}}

PAYING/TRIAL


```
{
    "licenseType": "PAYING",
    "useHostUnitWeighting": false,
    "expirationTime": 1539388799000,
    "suspensionType": "NONE",
    "syntheticEnabled": true,
    "chatEnabled": true,
    "maxAgents": 10,
    "maxPaasAgents": 15,
    "visitsQuota": 1000,
    "visitsAnnualQuota": 10000,
    "maxVisitsPerMinute": -1,
    "maxWebChecks": 125,
    "maxWebChecksAnnual": 1250,
    "sessionStorageQuota": 10000,
    "logAnalyticsStorageQuota": 9223372036854775807,
    "logAnalyticsIngressQuota": 9223372036854775807,
    "logAnalyticsIngressQuotaAnnually": 9223372036854775807,
    "symbolicationFileStorageQuota": 100,
    "hostUnitsQuota": 10,
    "maxHostUnitsQuota": -1,
    "overageEnabled": false,
    "externalApiQuota": 10,
    "iotEntityQuota": 20,
    "iotTsQuota": 10,
    "customMetricsLimit": 2,
    "customMetricsOverageLimit": 9223372036854775807,
    "overageCustomMetrics": true,
    "infrastructureSupportedTechnologies": {
        "pluginAgent": true,
        "networkAgent": true,
        "logAgent": true,
        "infrastructureOnlySupport": true,
        "maxInfrastructureOnlyAgents": 20
    },
    "retentionRum": 3024000000,
    "retentionWebcheck": 3024000000,
    "retentionService": 1209600000,
    "retentionCode": 864000000,
    "isConsumption": false,
    "isCreditExhausted": false,
    "isRumEnabled": true
}
```


Visit Retention in ms (=35 days)



License usage info not available through API

Creating an environment

Creating a new Environment

Create Tenant

POST https://{{ManagedURI}}/api/v1.0/control/tenantManagement/createTenant

- 2 Parts
  - TenantConfigDto
  - LicenseConfigDto

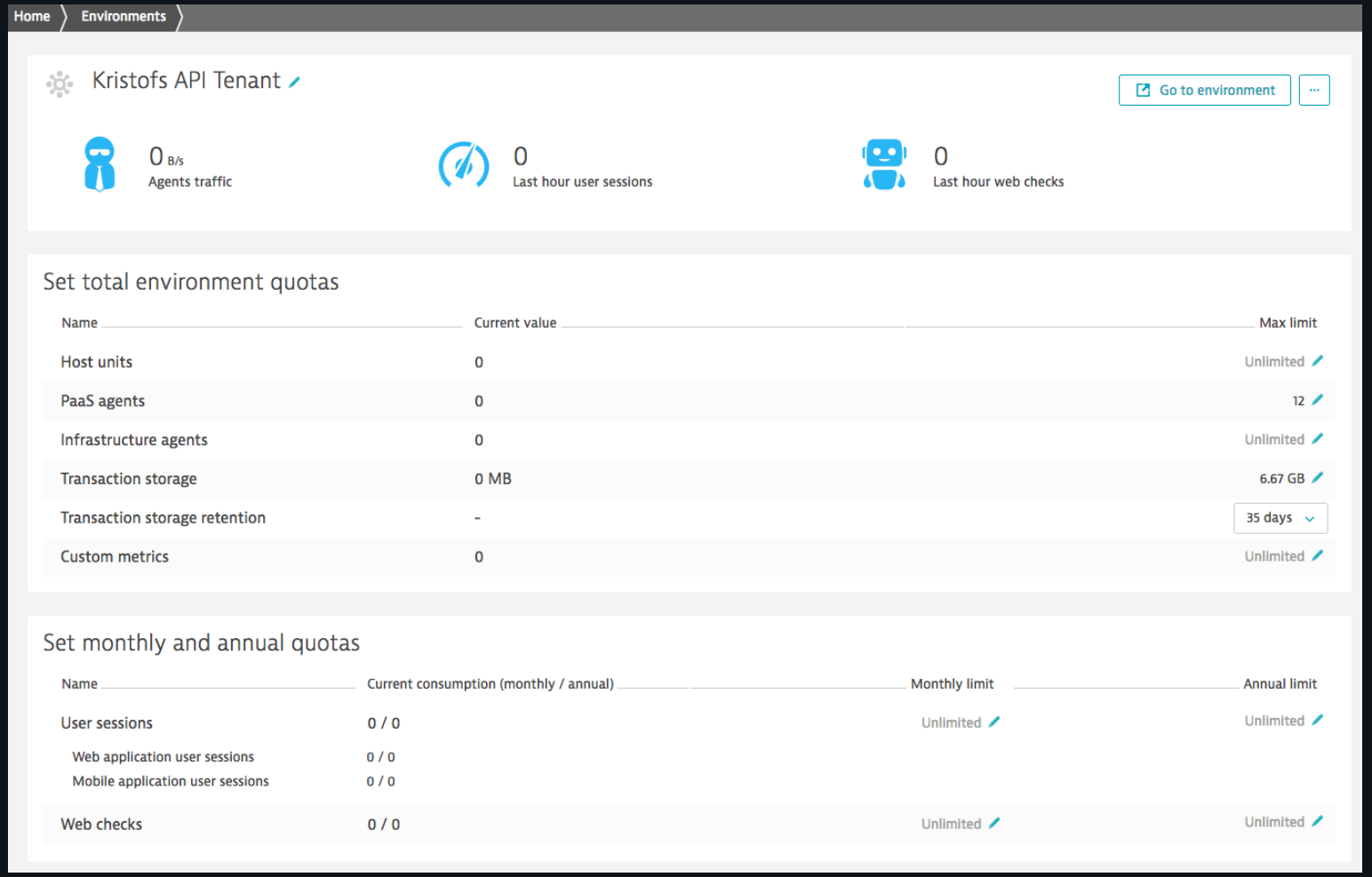

```
{
  "tenantConfigDto": {
    "alias": "mySecondEnvironment",
        "replicationFactor": -1,
        "domainNames": [],
        "buildUnits": [],
        "internalAlias": "mySecondEnvironment",
        "displayName": "mySecondEnvironment",
        "timezoneId": "UTC",
        "countryCode": "",
        "accountName": "",
        "accountId": -1,
        "blockedUI": false,
        "sfdcTenantId": null,
        "beaconDomain": "",
        "name": "mySecondEnvironment",
        "tenantUUID": "mySecondEnvironment",
        "isActive": true,
        "customerActivated": false
  },
```



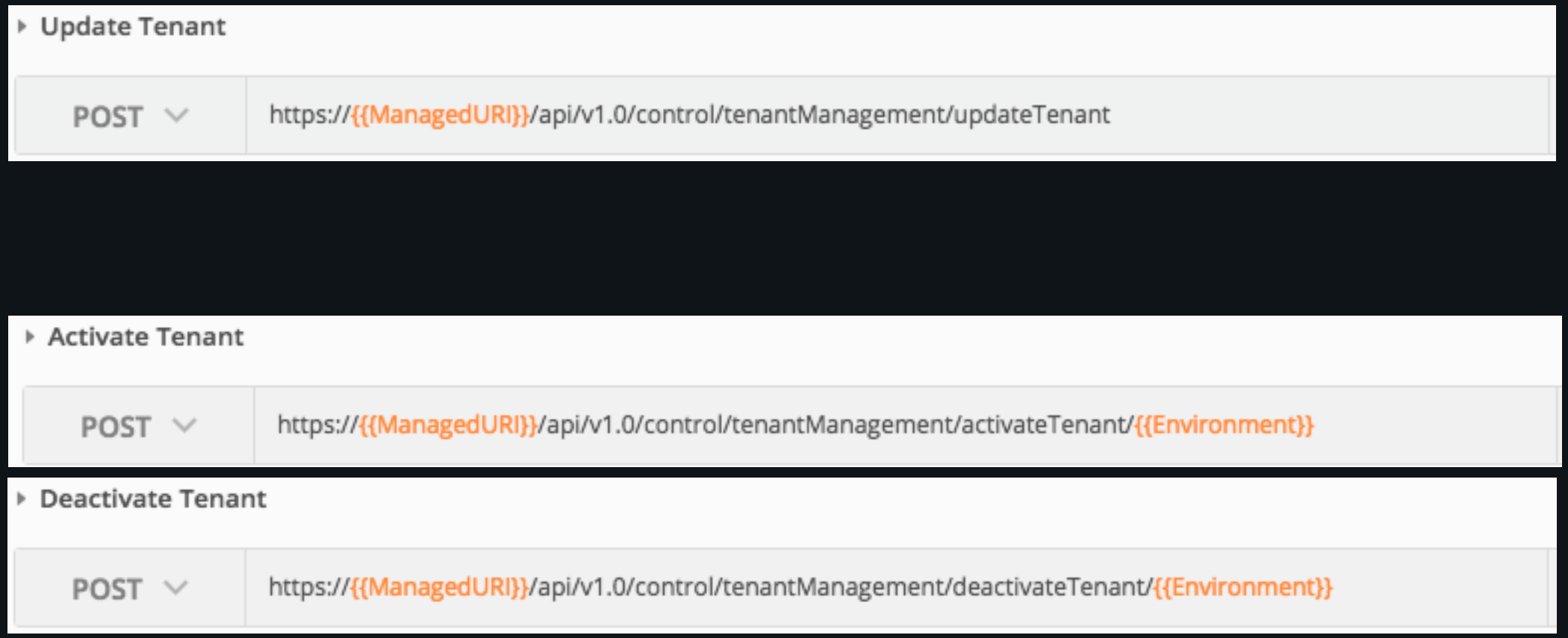
```
  "licenseDto": {
    "licenseType": "TRIAL",
    "useHostUnitWeighting": false,
    "expirationTime": 1539388799000,
    "suspensionType": "NONE",
    "syntheticEnabled": true,
    "chatEnabled": true,
    "maxAgents": 2147483647,
    "maxPaasAgents": 2147483647,
    "sessionStorageQuota": 2147483647,
    "logAnalyticsStorageQuota": 9223372036854775807,
    "logAnalyticsIngressQuota": 9223372036854775807,
    "logAnalyticsIngressQuotaAnnually": 9223372036854775807,
    "symbolicationFileStorageQuota": 100,
    "externalApiQuota": 10,
    "maxWebChecks": 9223372036854775807,
    "iotEntityQuota": 20,
    "iotTsQuota": 10,
    "hostUnitsQuota": 9223372036854775807,
    "maxHostUnitsQuota": -1,
    "overageEnabled": false,
    "infrastructureSupportedTechnologies": {
        "pluginAgent": true,
        "networkAgent": true,
        "logAgent": true,
        "infrastructureOnlySupport": true,
        "maxInfrastructureOnlyAgents": 9223372036854775807
    },
    "retentionRum": 3024000000,
    "retentionWebcheck": 3024000000,
    "retentionService": 1209600000,
    "retentionCode": 864000000,
    "isConsumption": false,
    "isRumEnabled": true,
    "isCreditExhausted": false
  }
```


Unique: Once created, cannot be re-created even when deleted!

Great Success!



Other Environment CRUD / (De)activate API Actions



Get/set Extended Config for OneAgent Auto Update Management

▸ Get Extended Config

GET https://{{ManagedURI}}/api/v1.0/control/tenantManagement/extendedConfig/{{Environment}}

▸ Set Extended Config

POST https://{{ManagedURI}}/api/v1.0/control/tenantManagement/extendedConfig/{{Environment}}

- Allows to see what the OneAgent update policy is for a given environment


```
{
    "autoUpdateConfig": {
        "agentAutoUpdate": false,
        "jsAgentAutoUpdate": true
    },
    "agentVersionConfig": {
        "standardAgentVersion": ""
    },
    "logAnalyticsConfig": {
        "linuxNfsSupport": false
    }
}
```


OneAgent updates

Automatically update all OneAgent instances

This setting can be overridden for individual hosts at Settings > Monitoring overview > Hosts

Standard version for all new hosts is: Always use latest version

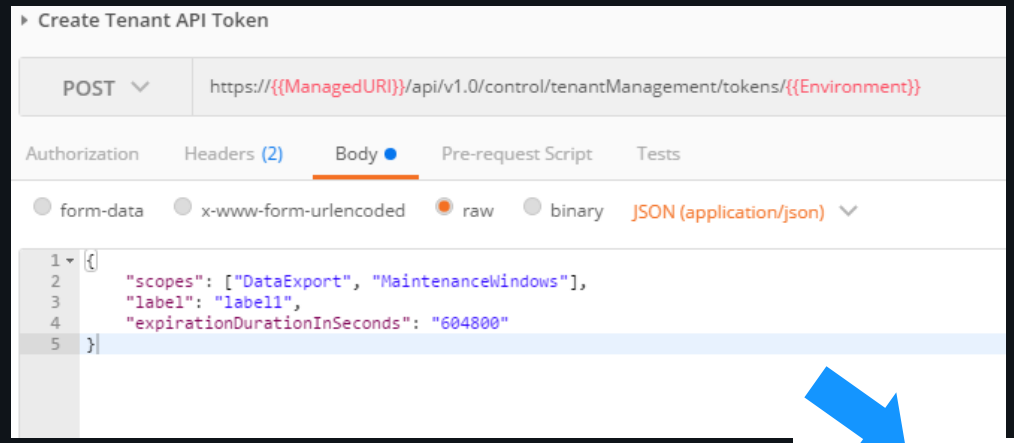
Note: The selected version is also used for PaaS integrations.

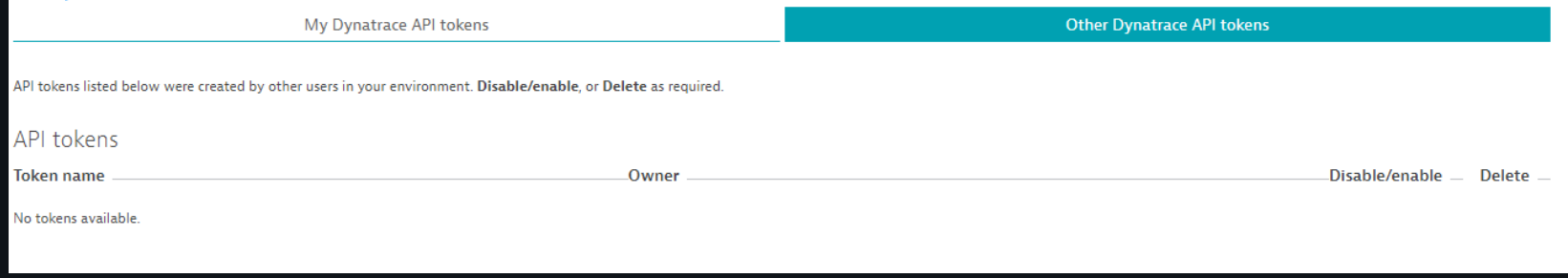
Automatically update real user monitoring JavaScript library

The real user monitoring JavaScript library is up-to-date.

Create Environment API Token

- Create API tokens with scope, label and Duration in Seconds (duration defaults to 7 days)
- Scope options (currently): DataExport, InstallerDownload, SupportAlert, DcrumIntegration, AdvancedSyntheticIntegration, PluginUpload, MemoryDump, MaintenanceWindows



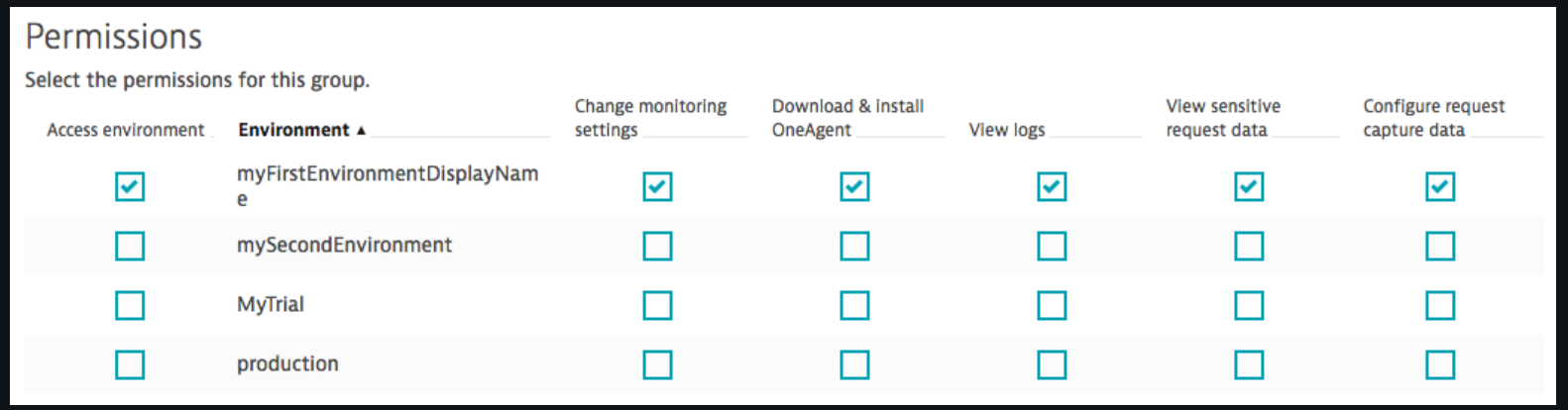


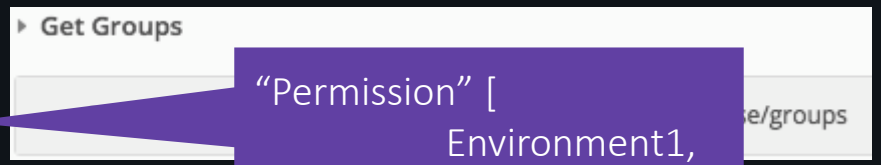
Managing Groups

Getting all groups

/<id> for a specific group


```
[
    {
        "id": "testgroup",
        "name": "testGroup",
        "ldapGroupNames": [
            "testGroup"
        ],
        "accessRight": {
            "VIEWER": [
                "myFirstEnvironmentUUID"
            ],
            "MANAGE_SETTINGS": [
                "myFirstEnvironmentUUID"
            ],
            "AGENT_INSTALL": [
                "myFirstEnvironmentUUID"
            ],
            "LOG_VIEWER": [
                "myFirstEnvironmentUUID"
            ],
            "VIEW_SENSITIVE_REQUEST_DATA": [
                "myFirstEnvironmentUUID"
            ],
            "CONFIGURE_REQUEST_CAPTURE_DATA": [
                "myFirstEnvironmentUUID"
            ]
        },
        "isClusterAdminGroup": false
    },
    {
        "id": "admin",
        "name": "admin",
        "ldapGroupNames": [
            "admin"
        ],
        "accessRight": {},
        "isClusterAdminGroup": true
    }
]
```

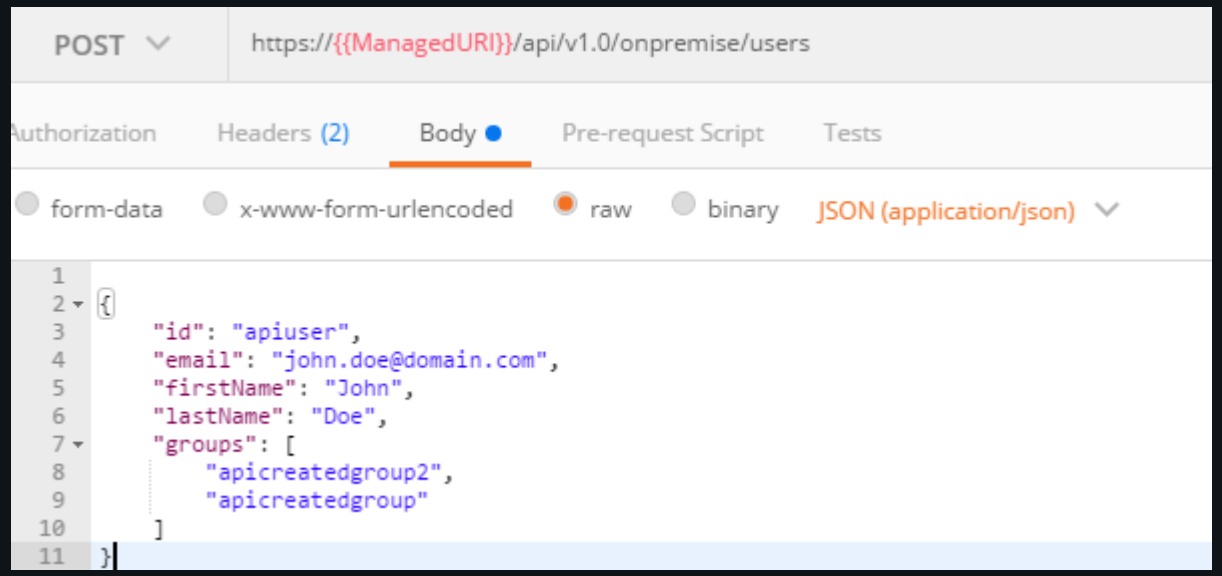


"Permission" [
Environment1,
Environment2
]

Getting all users

- https://{{ManagedURl}}/api/v1.0/onpremise/users

Creating a user



Also sends the invite email when creating via API

Command Line Interface

Dynatrace Command Line Interface (CLI)

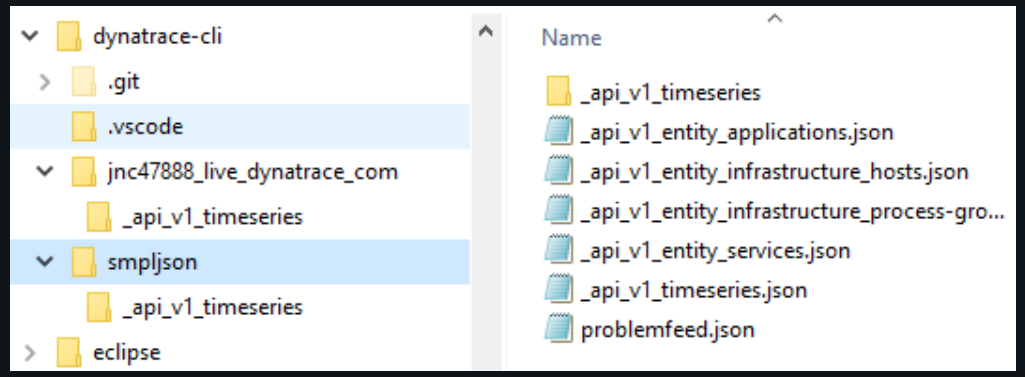
- Clone from https://github.com/Dynatrace/dynatrace-cli
- Interacts with Dynatrace API: https://help.dynatrace.com/dynatrace-api/
- Main Use Cases
  - Easy Query of Smartscape, e.g: All hosts with specific tag
  - Push Tags and Events on Smartscape Entities, e.g: Pipeline Deployment Events
  - Query Timeseries Data, e.g: from a Quality Gate in your Build Pipeline
  - Generate Reports, e.g: Load Testing or Business Reports
  - Pull Problem Details, e.g: For Auto Mitigation Actions
- Pre-Requisites
  - Python Runtime
  - Dynatrace Environment (SaaS or Managed) & Dynatrace API Token
- Feedback welcome!

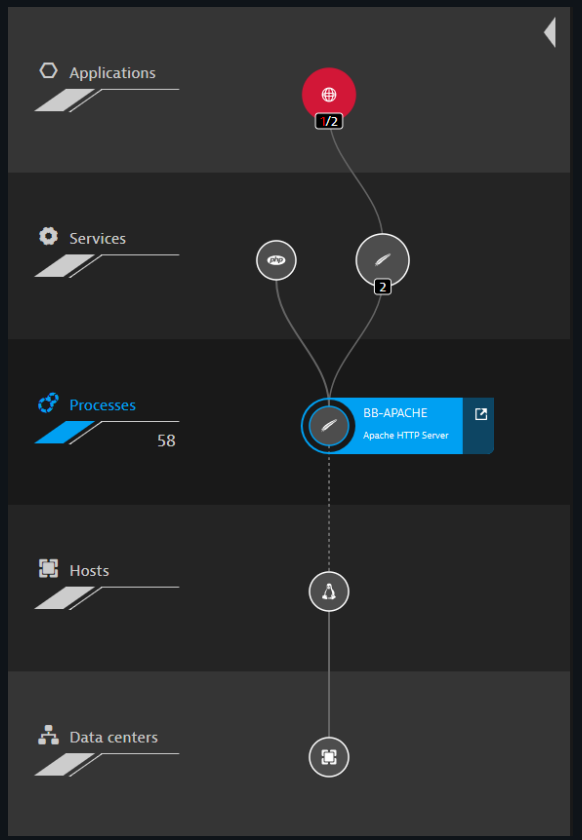
How the CLI works?

1. Read configuration from dtconfig.json
2. If using cache and current data is in cache read it from there
3. Use Dynatrace API to query data from LIVE environment
4. Persist latest query in cache
5. Perform CLI logic on data and return result to user

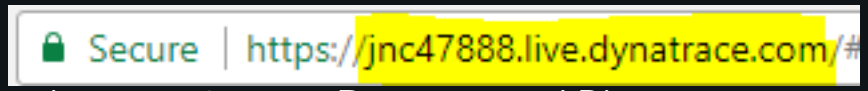
```
py dtcli.py ent app www.easytravel.com
```

5 1 2 4 3





Connecting to SaaS vs Managed vs “Demo Mode”

- Dynatrace SaaS Environment
  - Get your own SaaS Trial Environment: https://www.dynatrace.com/trial/
  - Environment URL 

  - API Token: Settings -> Integration -> Dynatrace API

- Dynatrace Managed
  - Get your Dynatrace Managed Trial: https://www.dynatrace.com/trial/managed/
  - Environment URL: 

  - API Token: Settings -> Integration -> Dynatrace API

- Demo Mode (Default On)
  - Environment URL: smpljson
  - API Token: not required!

Hands-On

Sample Entity Query Commands

BASIC Query Format

ent <type> <query> <resulttags>

List all hosts

```
ent host .*
```

Query list of OS Types

```
ent host .* osType
ent host .* osType displayName
```

Query hosts by name pattern

```
ent host .*apache.*
ent host displayName=.*apache.*
```

Query host by IP Pattern

```
ent host ipAddresses=172.31.6.* displayName
ent host ipAddresses=172.31.6.* displayName,hypervisorType
```

Retrieve all details of a particular host

```
ent host entityId=HOST-C155DC9790027845 *
```

Query a specific application

```
ent app .*easyTravel.*
```

Query all process groups

```
ent pg .*
```

Query processes of a specific Java Main Class

```
ent pg javaMainClass=com.mycompany.mymainclass
```

Make yourself familiar with the Entity JSON


```
[{
    "osArchitecture": "X86",
    "ipAddresses": ["52.202.165.203", "172.31.6
    "displayName": "BB1-apache-tomcatjms-iis"
    "entityId": "HOST-C155DC9790027845",
    "toRelationships": {
        "runsOn": ["PROCESS_GROUP-D715832732
        "isSiteOf": ["GEOLOC_SITE-95196F3C9A4F4215", "AWS_AVAILABILITY_Z
        "isNetworkClientOfHost": ["HOST-E096B0AF4B2C4276", "HOST-B208ACE2
    },
    "customizedName": "BB1-apache-tomcatjms-iis",
    "osType": "WINDOWS",
    "osVersion": "Windows Server 2008 R2 Service Pack 1, ver. 6.1.7601",
    "fromRelationships": {
        "isNetworkClientOfHost": ["HOST-AD521F83C70164D5", "HOST-C155DC97
    },
    "hypervisorType": "XEN",
    "discoveredName": "et-demo-2-win1"
}, {
```

Query Services by name or technology type

```
ent srv .* displayName
ent srv .* serviceTechnologyTypes
ent srv serviceTechnologyTypes=.* displayName, serviceTechnologyTypes
ent srv serviceTechnologyTypes=Java displayName, serviceTechnologyTypes
```

List of all service types

```
ent srv .* serviceType
```

Query all database services

```
ent srv serviceType=DATABASE displayName
```

Sample Entity Query Commands - Tags

BASIC Query Format for Tags or other nested Elements

```
listname/[context:][key][?valuekey]=value
```

Query hosts with a specific AWS Tag

```
ent host tags/AWS:Name=et-demo.*
```

More tag based queries

```
ent host tags/?value=et-demo.*
ent host tags/context#AWS:key#Name=et-demo.*
ent host tags/?key=Email
ent host tags/context#AWS?key=Email
ent host tags/key#Email?value=.*@dynatrace.com displayName

ent host tags/?key=OpenStack
ent host tags/context#CONTEXTLESS?key=OpenStack
```

Entity Object in JSON including Tags


```
[{
    "osArchitecture": "X86",
    "ipAddresses": ["52.202.165.203", "172.31.6.11
    "displayName": "BB1-apache-tomcatjms-iis",
    "entityId": "HOST-C155DC9790027845",
    "toRelationships": {
        "runsOn": ["PROCESS_GROUP-D71583273295BF32
        "isSiteOf": ["GEOLOC_SITE-95196F3C9A4F4215
        "isNetworkClientOfHost": ["HOST-E096B0AF4B2
    },
    "tags": [{
        "key": "OpenStack",
        "context": "CONTEXTLESS"
    }, {
        "value": "APL",
        "key": "Usage",
        "context": "AWS"
    }, {
        "value": "et-demo-2-win1",
        "key": "Name",
        "context": "AWS"
    }, {
        "value": "true",
        "key": "ShouldBeReserved",
        "context": "AWS"
    }, {
        "value": "DEMOABILITY",
        "key": "Category",
        "context": "AWS"
    }],
    "customizedName": "BB1-apache-tomcatjms-iis",
    "osType": "WINDOWS",
```


context?key=value

context:key=value

Sample Entity Query Commands – Other Metadata

Similar query not only works for tags

```
ent pg softwareTechnologies/?type=APACHE_HTTPD
ent pg softwareTechnologies/?type=PYTHON
ent pg softwareTechnologies/type#APACHE_HTTPD?version=2.*
```

Entity Object with more MetaData


```
[{
    "listenPorts": [4990, 433, 35347, 70],
    "entityId": "PROCESS_GROUP-5BDC2902F3F36B08",
    "softwareTechnologies": [{
        "type": "APACHE_HTTPD",
        "version": "2.4.18",
        "edition": null
    }, {
        "type": "PYTHON",
        "version": null,
        "edition": null
    }],
    "metadata": {
        "apacheConfigPaths": "/etc/apache*/apache*.conf"
    },
```


type#APACHE_HTTPD:edition#xyz?version=2.*

Sample Timeseries Queries

BASIC Timeseries Format

ts <list|describe|query|push> <timeseries>

List all available Timeseries

```
ts list
```

All Timeseries for a certain dimension

```
ts list dimensions=APPLICATION
ts list dimensions=HOST
ts list dimensions=SERVICE_METHOD
```

Timeseries matching a name

```
ts list .*response.*
Ts list .*response.* displayName
```

Describe a Timeseries

```
ts describe com.dynatrace.builtin:appmethod.useractionsperminute
```

Describe a built-in Timeseries

```
ts describe appmethod.useractionsperminute
```

Query for List of Key User Actions and Key Requests

```
ts queryent appmethod.useractionsperminute[count%hour]
ts queryent servicemethod.responsetime
```

BASIC Query Format

```
ts query timeseries[aggregation%timeframe]
Aggregation: needs to be provided by Timeseries
Timeframe:
-  hour,2hours,6hours,day,week,month : Relative Time from NOW()
-  60, 120, 150 … : Relative Minutes back from NOW()
-  1503951000000:   Absolute Full Timestamp
-  LOADTEST1:       Absolute Timestamp from Event (TODO!)
```

Query Basic Timeseries

```
ts query host.cpu.system[avg%hour]
ts query host.cpu.system[max%2hours]
ts query host.cpu.system[avg%300],host.cpu.system[max%300]
ts query host.cpu.system[avg%hour] HOST-AD521F83C70164D5
```

Query for Key User Actions and Key Requests Time Series

```
ts query appmethod.useractionsperminute[count%hour]
ts query servicemethod.responsetime SERVICE_METHOD-98E2E242C96F31E9
```

https://help.dynatrace.com/dynatrace-api/timeseries/how-do-i-fetch-the-metrics-of-monitored-

Sample Reporting: DQLR (Dynatrace Query Language)

BASIC DQL/DQLR Format

```
dql|dqlr <entitytype> <entity> <metrics>
entitytype: app | appmethod | servicemethod | srv | pg | host
entity:     entityname
metrics:    metricname[aggr%time],metricname[aggr%timefrom:timeto]
```

Raw JSON Report Data, e.g: CPU Utilization of All Hosts

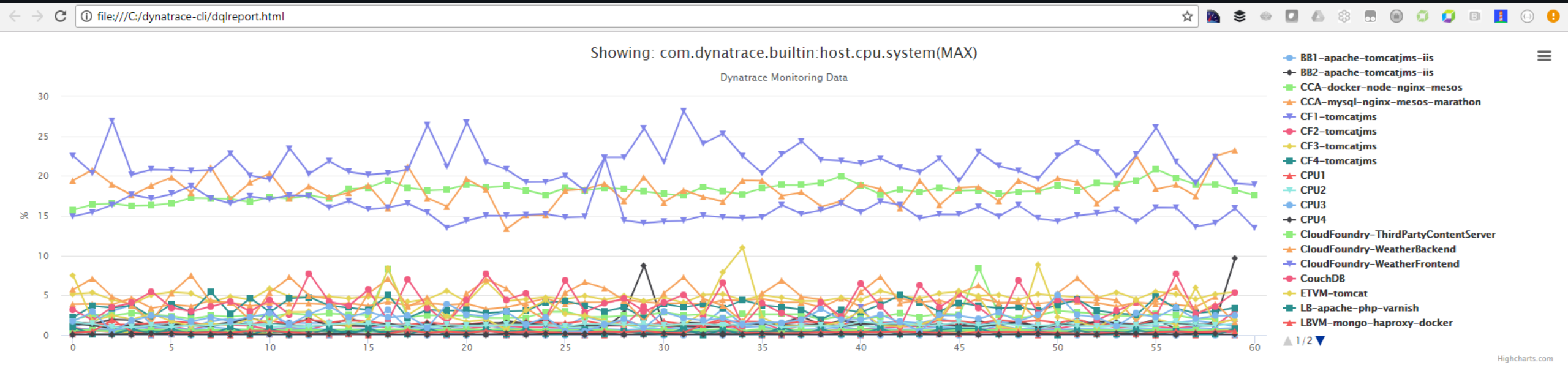
```
dql host .* host.cpu.system[max%hour]
```

Generate Report on DQL, e.g: CPU Utilization of All Hosts or AWS Hosts

```
dqlr host .* host.cpu.system[max%hour]
dqlr host tags/?context=AWS host.cpu.system[avg%day]
```

Report on Key Service Request

```
dqlr servicemethod checkCreditCard servicemethod.responsetime[avg%hour]
```



Sample Deployment Events

BASIC Event Format

```
evt <query|push> <entity> <options>
```

Query Events of all or specific entities in a certain timeframe

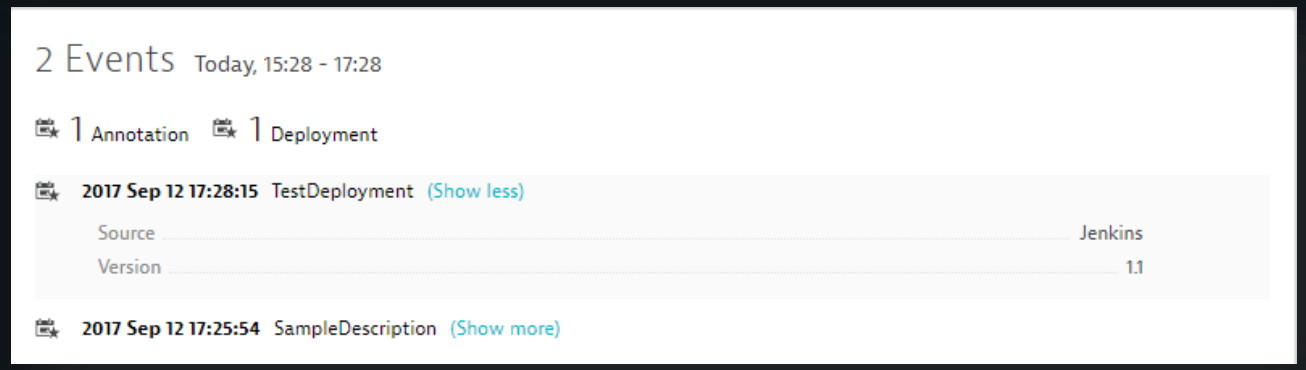
```
evt query
evt query entityId HOST-776CE98524279B25
evt query host .*demo.*
evt query eventType=SERVICE_RESPONSE_TIME_DEGRADED from=60 to=0
```

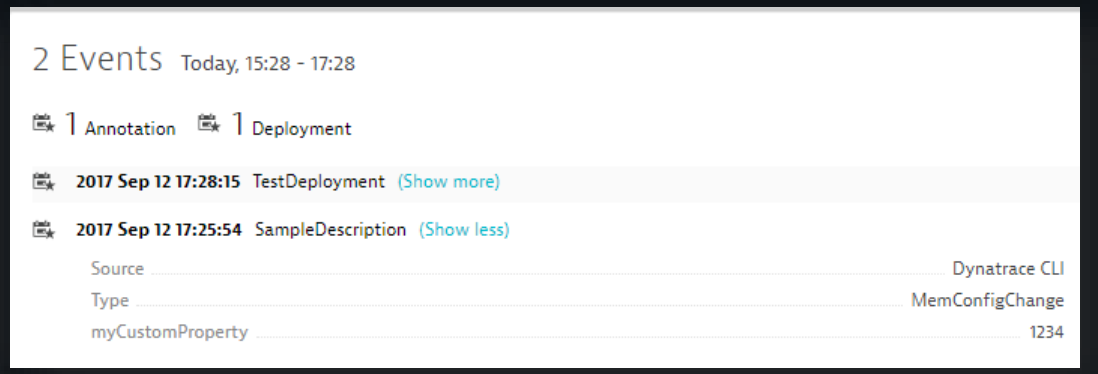
Push Deployment Events

```
evt push host .*demo.* deploymentName=TestDeployment deploymentVersion=1.1 source=Jenkins
evt push host .*demo.* deploymentName=TestDeployment deploymentVersion=1.2 source=Jenkins ciBackLink=yourlink start=20 end=5
```

Push Deployment Annotations

```
evt push host HOST-776CE98524279B25 eventType=CUSTOM_ANNOTATION annotationType=MemConfigChange
annotationDescription=Changed%20Memory%Settings myCustomProperty=1234
```





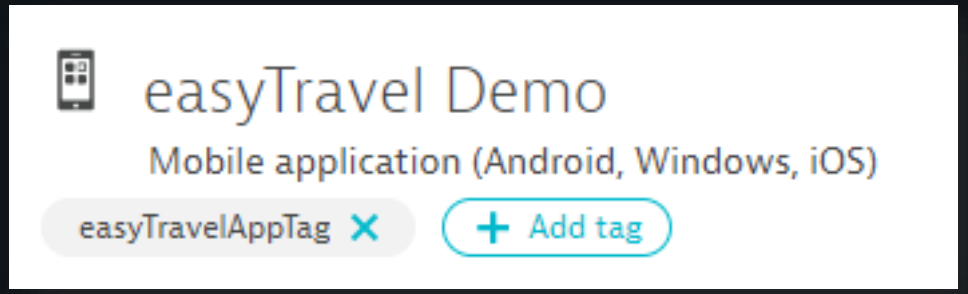
Sample Tags

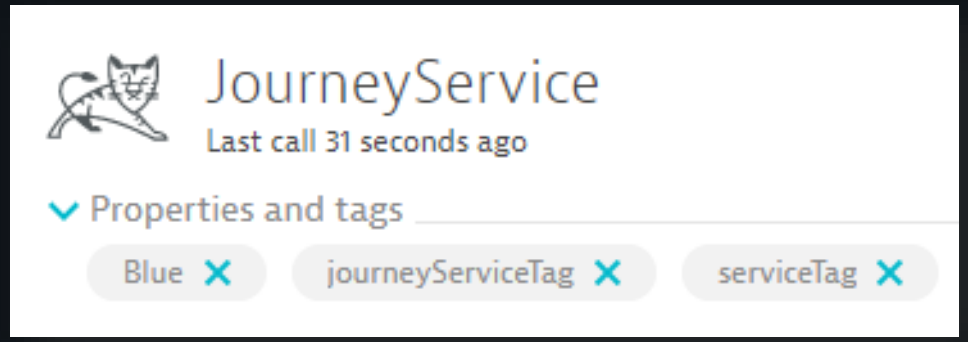
BASIC Event Format

```
tag <entityid|entityquery> <list of tags>
```

Publish some Tags

```
dtcli tag app .*easyTravel.* easyTravelAppTag
dtcli tag srv JourneyService journeyServiceTag,serviceTag
dtcli tag app entityId=APPLICATION-F5E7AEA0AB971DB1 easyTravelAppTag
```





Upcoming Features

- Query and Update Problems
  - Get all problem details PLUS deployment events on correlated entities
  - Pull and Push comments
- Push Custom Metrics
  - Timeseries API allows you to define custom external metrics
  - Lets make it available via CLI
- Maintenance API
  - Query and Define Maintenance Windows via CLI
- Leverage more of the new REST API Smartscape Filter Options
- Historical Caching of Smartscape and Comparison
  - E.g: Show me the change of “MyService” Instances over the last 6 hours

Auto-remediation

Auto-remediation

- Also known as: self-healing, auto-mitigation, etc…
- Certain problems may be common or well-understood and require simple manual interaction to resolve them (e.g. rolling back a bad commit or restarting a server low on resources)
- Instead of waking up at 2AM to click a button, use Dynatrace in combination with your automation tools (e.g. Ansible Tower, Jenkins, Serverless functions, etc…) to recover automatically
- Dynatrace problem notifications and the API can provide you with valuable information to use in scripts that can take corrective actions
- There are several built in notifications and the custom integrations can be used in other cases

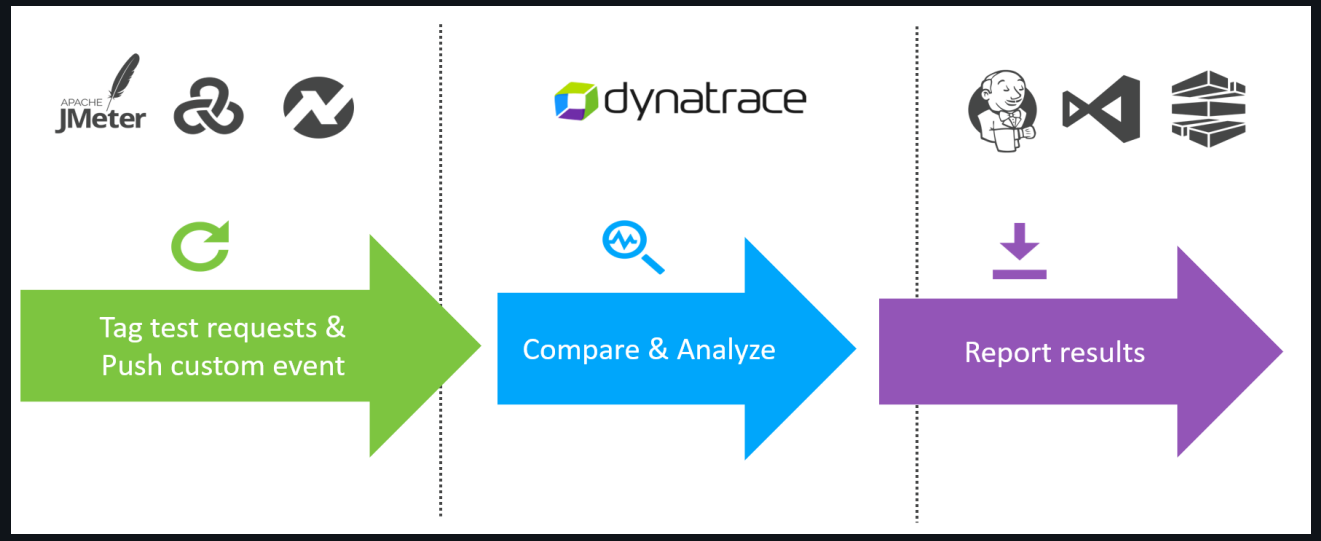
Auto-remediation example scenario

1. Dynatrace detects problem at 5PM on Friday tied to a new deployment event
2. A problem notification kicks off a script or playbook that attempts to resolve the issue and rolls back to the previous build if unable to do so
   - Restart crashed processes
   - Remove and replace bad nodes
   - Etc…
3. Automation tool comments on what action was taken on the problem card in Dynatrace
4. If any problems encountered the problem can be escalated to notify a human to take a look

Load Testing Integration

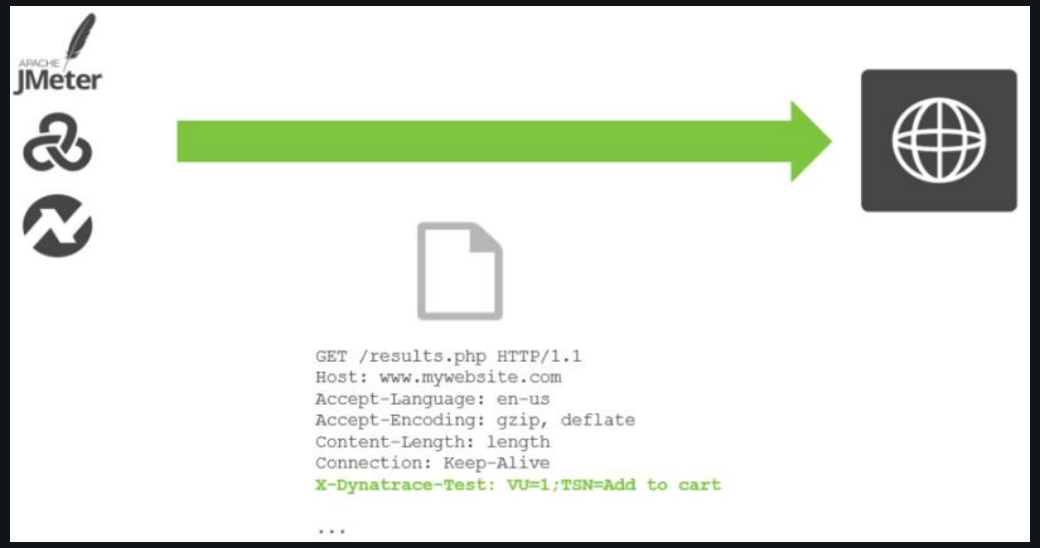
Integrate Dynatrace with load testing tools

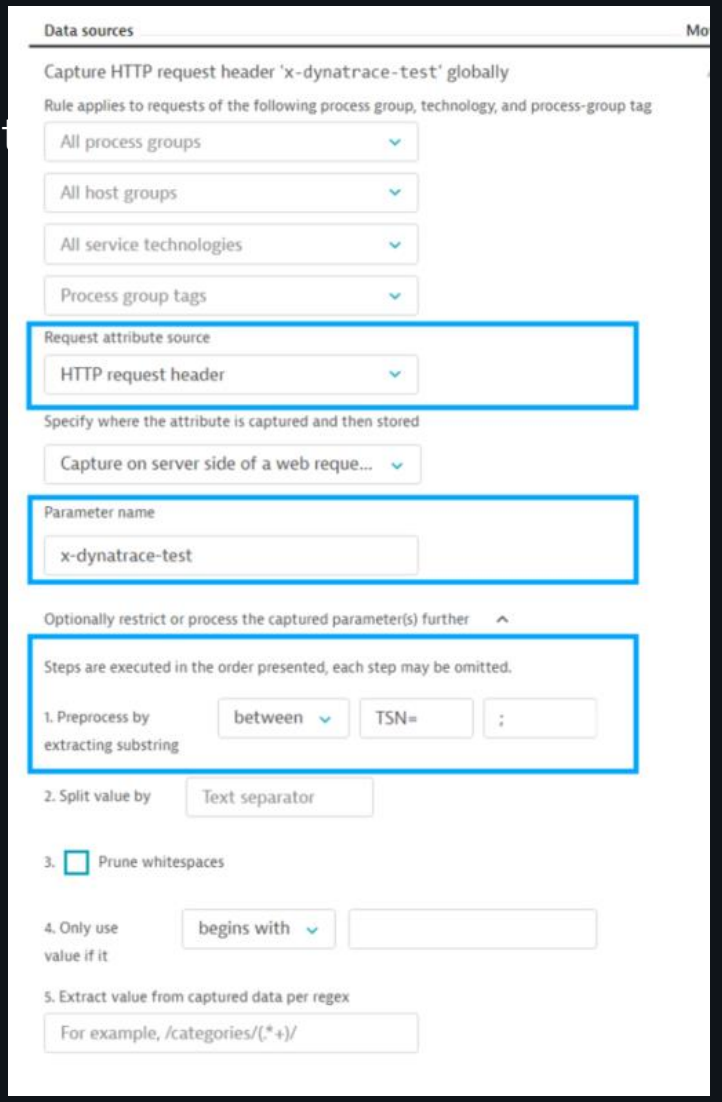
- Possible to stop broken builds using Dynatrace data!
- Dynatrace can help throughout the delivery pipeline:
  - Tagging tests and pushing deployment events
  - Comparing and analyzing tests
  - Reporting results to other tools (e.g. Jenkins, Amazon CodeDeploy, etc...)



Tagging your tests

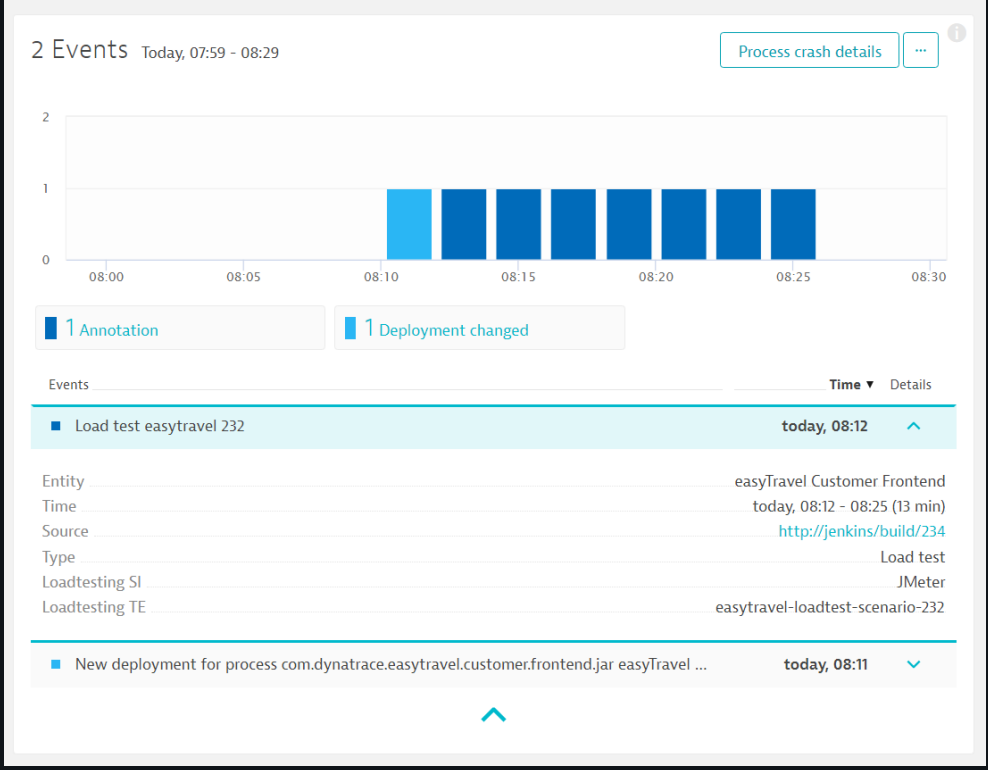
- Use your testing tool to tag requests and create corresponding request attributes in Dynatrace
  - Script name, test step name, virtual user ID, etc…
- Information will be extracted and applied to the monitored transactions for use in filtering and charting

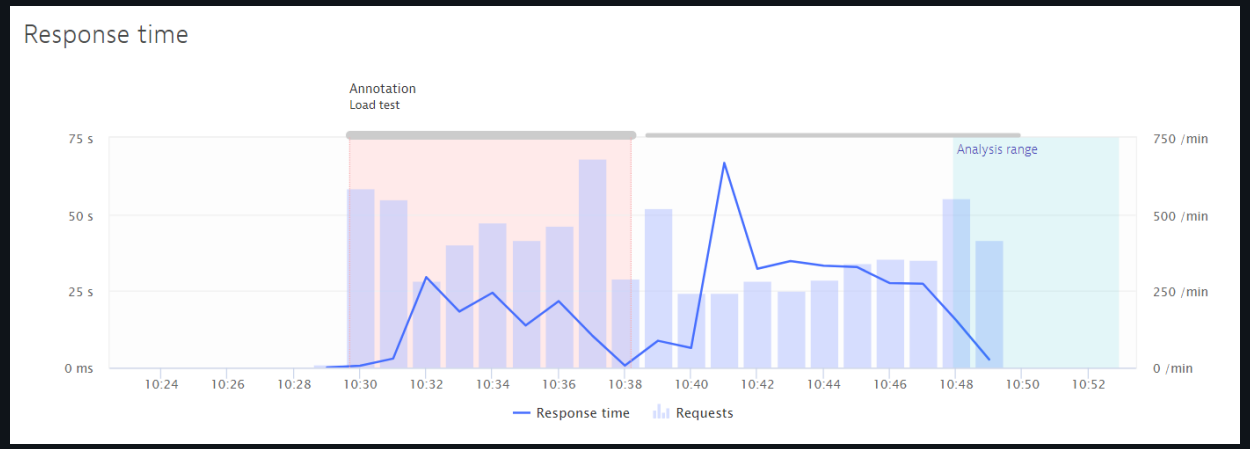




Pushing events and other test data

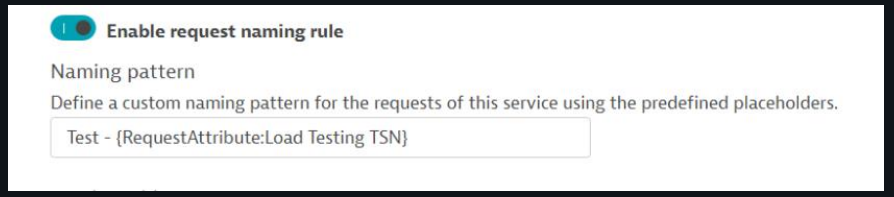
- Use the events API to push events and inform Dynatrace when your test is underway
- Use the custom metrics API to push other metrics from your testing tool (e.g. throughput, user load, etc…)

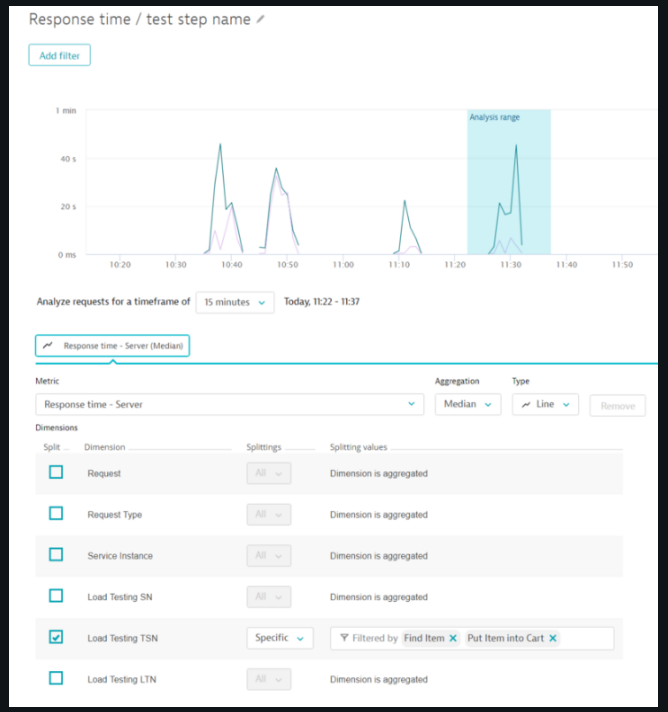


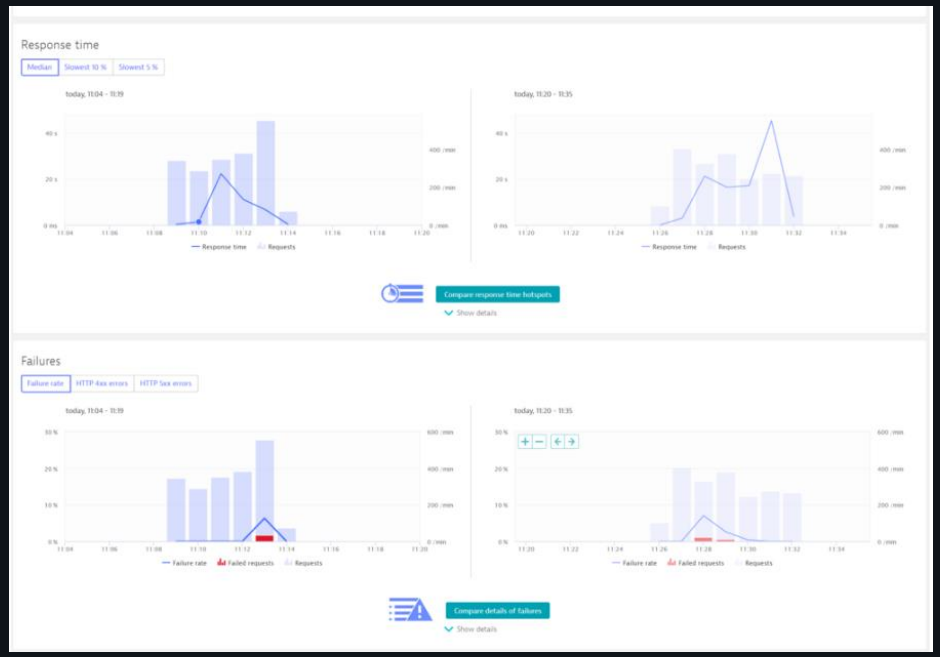


Analyzing your tests

- Many options exist for analyzing tests depending on the need
- Use your request attributes to filter analysis views and even customize request naming
- Create multidimensional analysis charts to monitor your tests over time
- Use comparison views to compare 2 different tests







Additional considerations

- You can use the Timeseries and Problem APIs to allow other tools in your pipeline to access and make use of Dynatrace data
  - Pass or fail builds in the deployment pipeline based on Dynatrace data
- Consider how maintenance windows should be used:
  - If testing in a production environment set a maintenance window during the test so that your baselines are not affected
  - If you have a dedicated testing environment don't use maintenance windows during tests so that you can leverage the AI-powered problem detection

LoadRunner Integration

- To help our LoadRunner users with the tagging of their requests, we've released the LoadRunner Request Tagging tool
- This tool enables you to patch your existing LoadRunner Virtual User Generator (VuGen) scripts by adding the x-dynatrace-test HTTP header and automatically populating it with your request attributes
- https://github.com/Dynatrace/Dynatrace-LoadRunner-Request-Tagging/releases

LoadRunner Integration

- Record a LoadRunner Virtual User Generator (VUGEN) script and adapt it as needed
- Download the latest version of the LoadRunner Request Tagging tool from GitHub
- Open your command prompt and patch your LoadRunner script
  - The LoadRunner Request Tagging tool uses the following syntax (more details can be found in the documentation):
    - java -jar DynatraceLRConverter.jar insert|delete -path <path_to_directory> | -header <path_to_h> -body <path_to_c>
- After running the tool, you can verify that the method addDynatraceHeaderTest has been added to your scripts before some of the key methods

LoadRunner Integration

- The HTTP header **"x-dynatrace-test"** is populated by the LoadRunner Request Tagging tool with the key/value pairs listed below
- These values can be captured by Dynatrace by defining request attributes.

TSN	Test Step Name	Logical test step within your load test script
LSN	Load Script Name	Name of the load test script
LTN	Load Test Name	Unique test execution identifier
VU	Virtual User	ID of the unique user who sent the request
PC	Page Context	Provides information about the document that is loaded in the currently processed page
SI	Source ID	Identifies the product that triggered the request

Questions?

Simply smarter clouds