Extending Dynatrace

Dynatrace Training Module

dynatrace

Agenda

- OneAgent SDK
- OpenKit
- Mobile SDK
- Plugins
- Remote Extensions
- W3C Tracing

OneAgent SDK

OneAgent SDK

- Manually instrument you application to extend end-to-end visibility for frameworks and technologies that are not yet supported by a dedicated code module
- Provides full access to all functionality (e.g. auto-baselining and AI-powered RCA)
- SDK is open source and in beta status
- Supported languages:
  - Java
  - Node.JS
  - C /C++
  - Python (EAP)
  - .NET (EAP)
  - Queue and Messaging (EAP)
- Requires changes to application source code and a OneAgent running on the host

OpenKit

RUM

Web

Mobile Web

Mobile

OpenKit

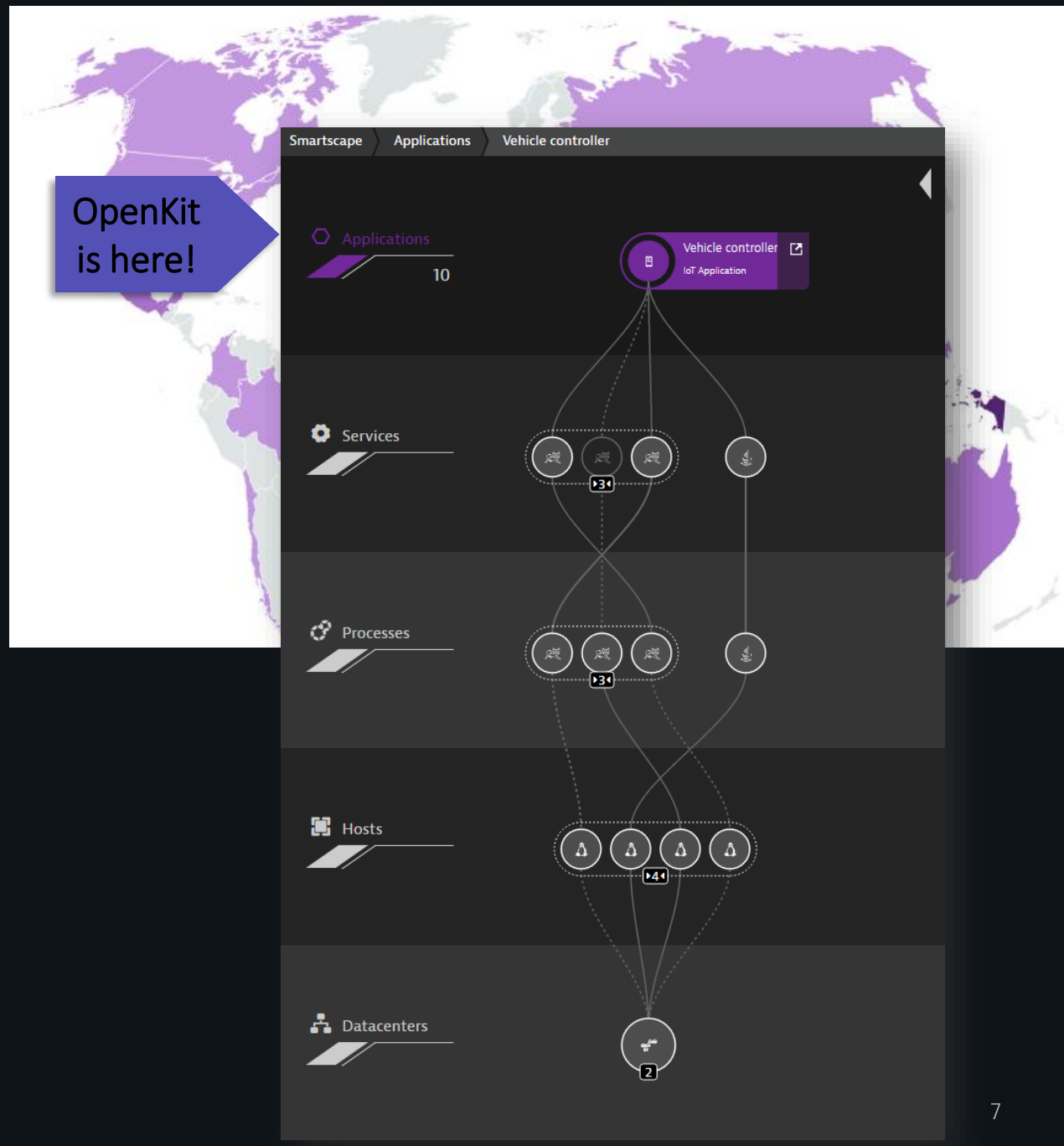
Rich Clients

Other Digital Touchpoints
TV, car, kiosk, dishwasher...

OpenKit: IoT applications

Capabilities

- Java & .NET reference implementation of RUM protocol
- Monitor **millions of devices and users**
- World map of **global distribution** of device activity
- Cohort analysis of **device types and properties**
  - platform, resolution, power level, firmware version, etc.
- Device **operation/user action reporting**
  - timings and durations
  - errors
  - reporting of nested operations
- Tagging of network requests to follow a **PurePath into backend**
- **Crash reporting** with stack traces and logs
- **Session tracking**



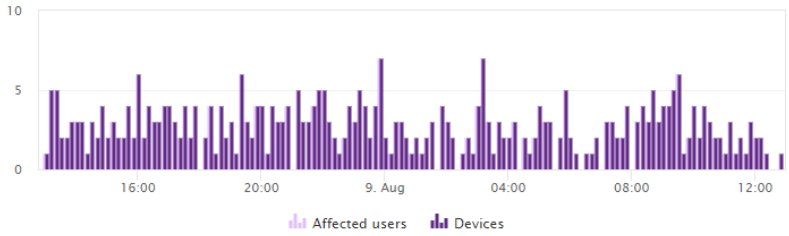
Code-level visibility

"We need to get stack traces, logs and device configurations to fix software related issues."

Applications > easyTravel Demo

Crashes and affected devices

Compares total number of crashes with numbers of affected devices during selected time period.



Crashes

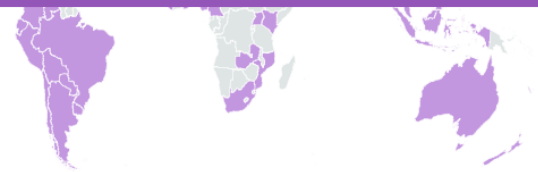
Crash report details are available back until **Jul 04 2017, 21:20**.

Filter various properties:

Start typing to filter...

- App Version
- Device
- OS version
- Platform
- Screen resolution

	Last occurrence	Total crashes	Impacted users	Details
	Today, 11:38	31	31	
	Today, 12:50	41	41	
cted 88 NullPointerException	Today, 12:16	63	62	
UIImage.swift:74 NSDemoException	Today, 10:32	17	17	
ProblemDetailFragment.onCreate:113 RuntimeException caused by NullPointerException	Today, 11:59	68	67	
CoreFoundation:1202432 NSRangeException	Today, 09:36	9	9	
ProblemDetailFragment.onCreate:120 IndexOutOfBoundsException	Today, 12:22	151	147	



Countries	Active sessions	Devices	Crash count
United States	2k	2k	185
China	360	345	31
Japan	255	248	14
Germany	147	145	6
United Kingdom	138	138	10
Republic of Korea	117	115	16
France	91	91	7
Canada	82	81	7
Italy	70	70	7
Netherlands	67	66	9

1 2 3 ... 9 10

Performance monitoring

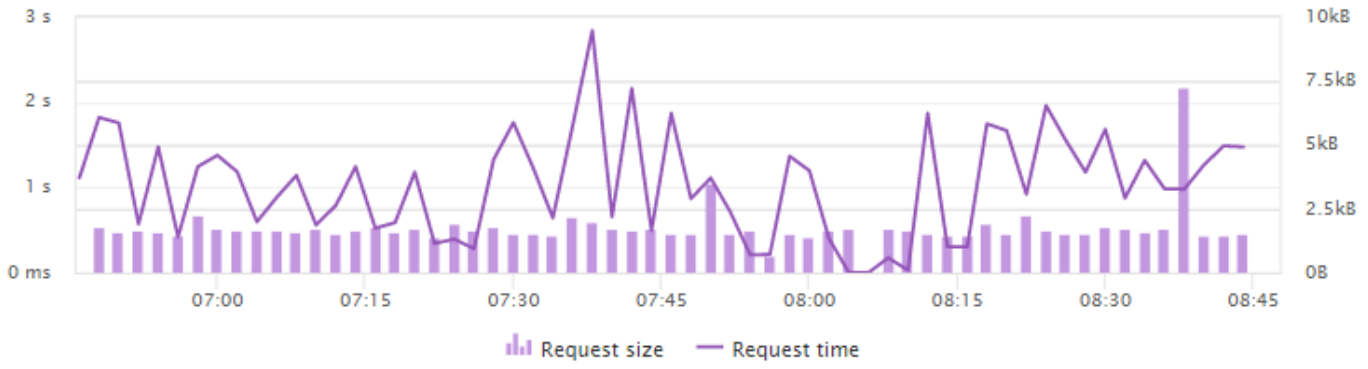
- Is the new firmware causing some communication troubles?
- Is the new firmware using more bandwidth than the previous versions?
- Are some backend services more prone to errors?

Top HTTP requests

Name	Requests	Request time	Error rate	Details
deviceconfig.buildingcontrol.com	7.50 /min	6.22 s	1.78 %	
billing.buildingcontrol.com	1.73 /min	586 ms	17.31 %	
customerprofile.buildingcontrol.com	1.33 /min	2.49 s	0.00 %	
facebook.com	0.60 /min	5.34 s	11.11 %	
weatherforcast.weatherservice.com	0.37 /min	6.86 s	9.09 %	

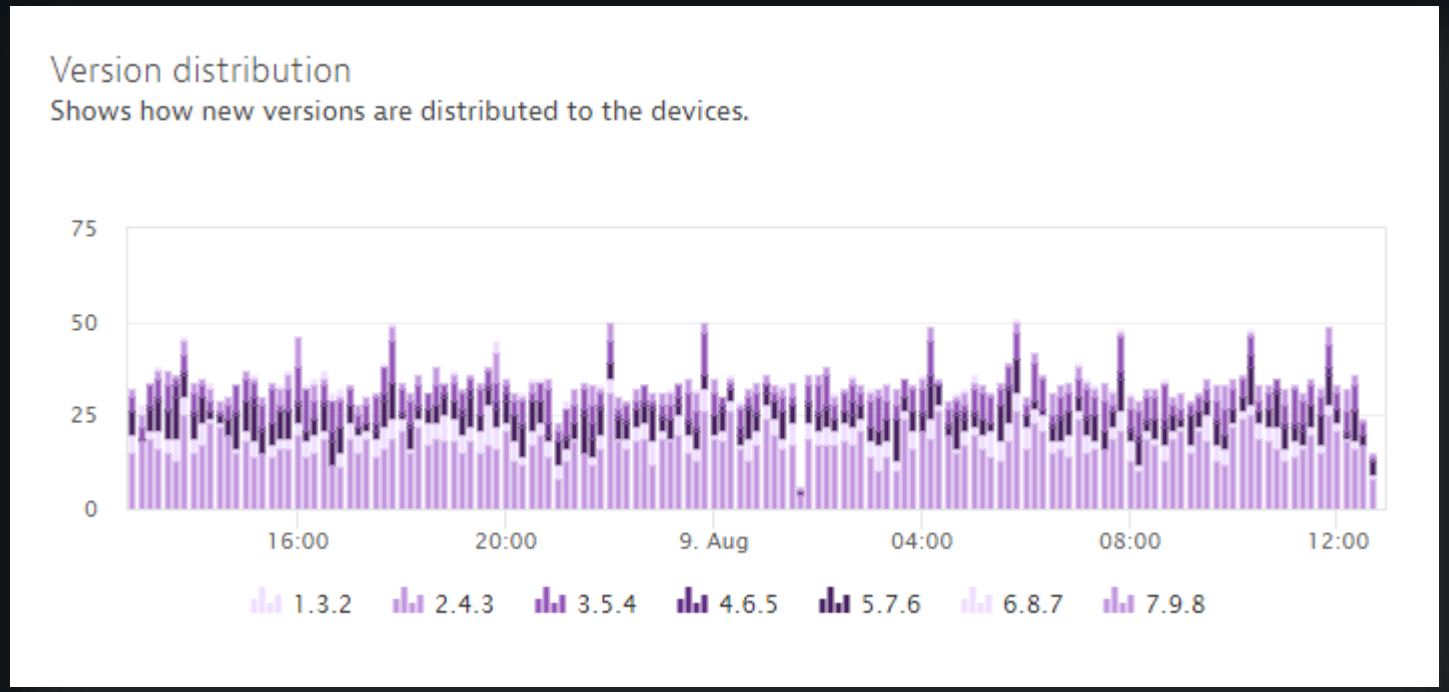
HTTP request size and performance

Compares the HTTP request size with client-side performance during selected time frame.



Firmware distribution

"We are operating millions of smart home gateways, distributed across all geographic regions, running different firmware versions, that are connected with our service backend infrastructure."



- Is my new firmware version as stable as the ones before?
- How many devices of a specific firmware version are currently online?
- How to get immediate notification if a firmware version does not work out as expected?

Availability

- Currently available for
  - Java (1.6+): https://github.com/Dynatrace/openkit-java/
  - .NET (3.5+, Core 1.0+): https://github.com/Dynatrace/openkit-dotnet/
  - more to come …
- Can be used for Dynatrace SaaS and Managed

What can/can't OpenKit do?

- What it can do:
  - create sessions, user actions and child user actions
  - report values, events, errors and crashes
  - trace web requests (to server-side PurePaths) including the request size
  - tag sessions with a username
- What it cannot do:
  - create server-side PurePaths (there is an SDK for that)
  - create metrics (there is an API for that)

SDK vs OpenKit

SDK

- Appears as a process and a service
- OneAgent required = FullStack (Network, Logs…)
- Each appearance has an entity in the SmartScape
- Scales to approx. 50k, goal 100k hosts

OpenKit

- Appears as an application
- RUM (User Actions, User Behavior, conversion goals…)
- One appearance as an Application
- Scales to 3000+ UA/s

Add a custom application

Docker

Manage

Deploy Dynatrace

Deployment status

Settings

Digital touchpoint monitoring

Beta

By providing you with a set of open source libraries, OpenKit enables you to instrument the digital touchpoints in your environment that aren't detected automatically by OneAgent. This means you can monitor the usage, performance, and user sessions of traditional rich-client applications, smart IoT applications, Alexa skills, and much more.

Create custom application

Monitor a custom application

Type in a name for the custom application you want to monitor. Once your custom application is created, you'll get detailed instructions on how to instrument it with OpenKit.

For example: My custom application

Note: It's not necessary to create a custom application if you only need to monitor a web application or a mobile app. Dynatrace OneAgent supports monitoring of these application types out-of-the-box.

Select one of the following icons that represents your application in the UI.

Cancel

Create custom application

Java | .NET | C++ | JavaScript | Alexa Skills

Beacon URL:

https://bf79563wis.bf-dev.dynatracelabs.com/mbeacon

Application ID:

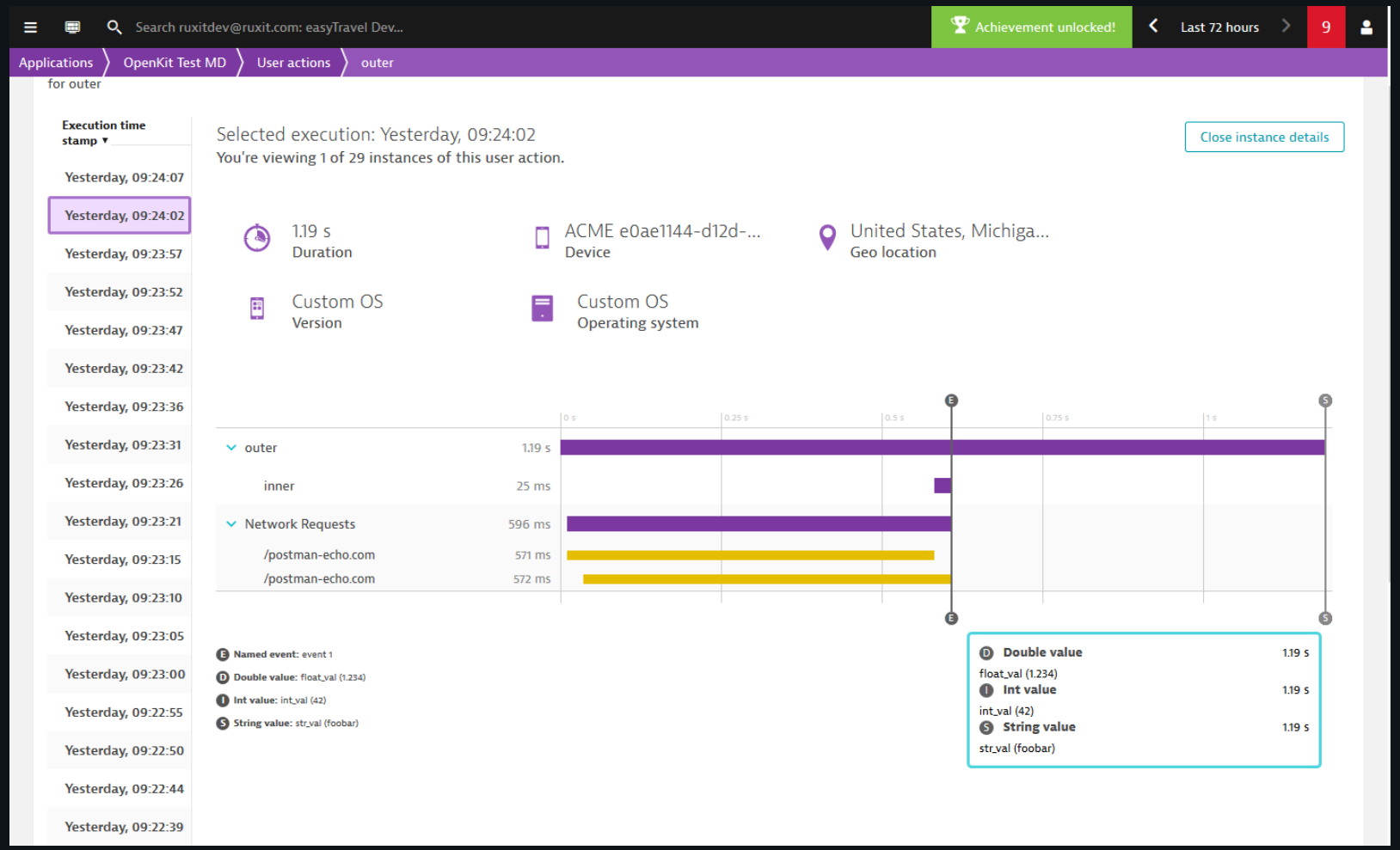
73c8adc8-769f-4b9a-a583-921c5f12c0f1

Beacons

If you don't receive the data you've expected, analyze incoming beacons

View incoming beacons

What does it look like in Dynatrace?



Quick look at the API

com.dynatrace.openkit.api

Interface OpenKit

```
public interface OpenKit
```

This interface provides basic OpenKit functionality, like creating a Session and shutting down the OpenKit.

Field Summary

Fields

Modifier and Type	Field and Description
`static java.lang.String`	`WEBREQUEST_TAG_HEADER` Name of Dynatrace HTTP header which is used for tagging web requests.

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
`Session`	`createSession(java.lang.String clientIPAddress)` Creates a Session instance which can then be used to create Actions.
`Device`	`getDevice()` Returns the Device used by this OpenKit instance.
`void`	`initialize()` Initializes the OpenKit, which includes waiting for the OpenKit to receive its initial settings from the Dynatrace/Appmon server.
`void`	`setApplicationVersion(java.lang.String applicationVersion)` Defines the version of the application.
`void`	`shutdown()` Shuts down the OpenKit, ending all open Sessions and waiting for them to be sent.

Concepts and Terminology

- **OpenKit:** represents an instance of the OpenKit, needed to create Sessions
- **Device:** represents the device and information about it
- **Session:** represents a Session that can contain Actions
- **(Root)Action:** represents timed Actions (up to 2 levels) that can contain key-value pairs, errors, …
- **WebRequestTracer:** provides the ability to trace a web request to another tier
- **Errors & Crashes:** represent handled/unhandled errors in the application
- **Identify Users:** provides the ability to identify users

Mobile SDK

Mobile SDK

- Report additional details about user sessions in your mobile apps
- Allows you to:
  - Create customer user actions
  - Measure web requests
  - Report errors
  - Tag users
- Available for iOS and Android
- Privacy settings can also be dynamically adjusted to ensure compliance with regs such as GDPR

OneAgent SDK for iOS

- Report additional details about mobile user sessions in your iOS app
- SDK is available automatically once the Dynatrace CocoaPod is added to your project
- Can be used in Swift as well as Objective-C
- Creating custom user action in:

```swift
// start action "search"
let action = DTXAction.enter(withName: "search")

// ...do some work here...

// end action "search"
action.leave()
```

Swift

```objc
// start action "search"
DTXAction *action = [DTXAction enterActionWithName:@"search"];

// ...do some work here...

// end action "search"
[action leaveAction];
```

Objective-C

OneAgent SDK for Android

- Report additional details about the mobile user sessions in your Andriod app
- You can use either the Dynatrace Gradle plugin or the command line to add the SDK
- Creating custom user action:

```
// start user action
DTXAction searchAction = Dynatrace.enterAction("search");

// ...do some work here...

// end the action after the search completed
searchAction.leaveAction();
```

Monitored Technologies (OneAgent Plugins)

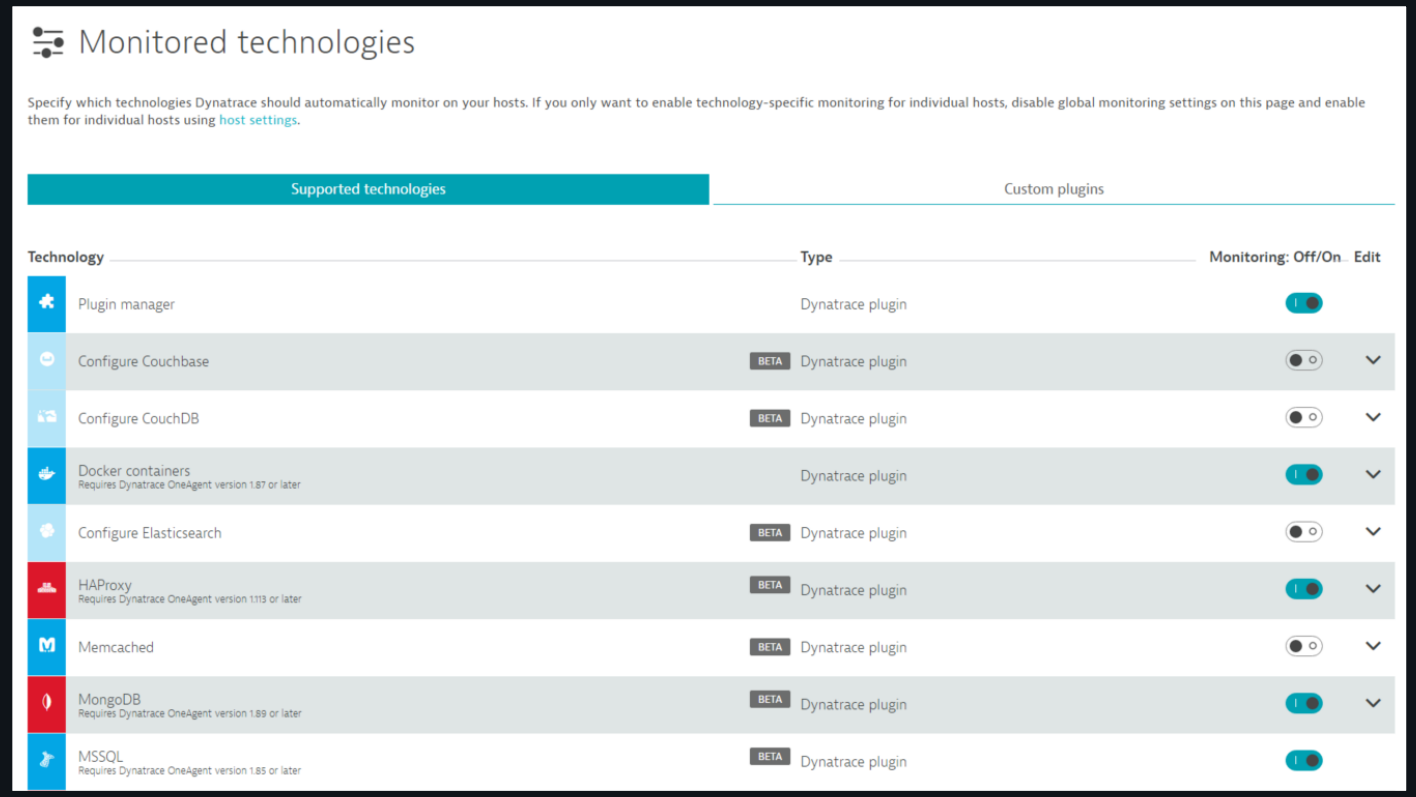
Monitored Technologies

- The Monitored Technology page is where you specify which technologies Dynatrace should monitor
- This is where both Built-in and Custom Plugins are configured
- You can configure them at the global or at the Host level

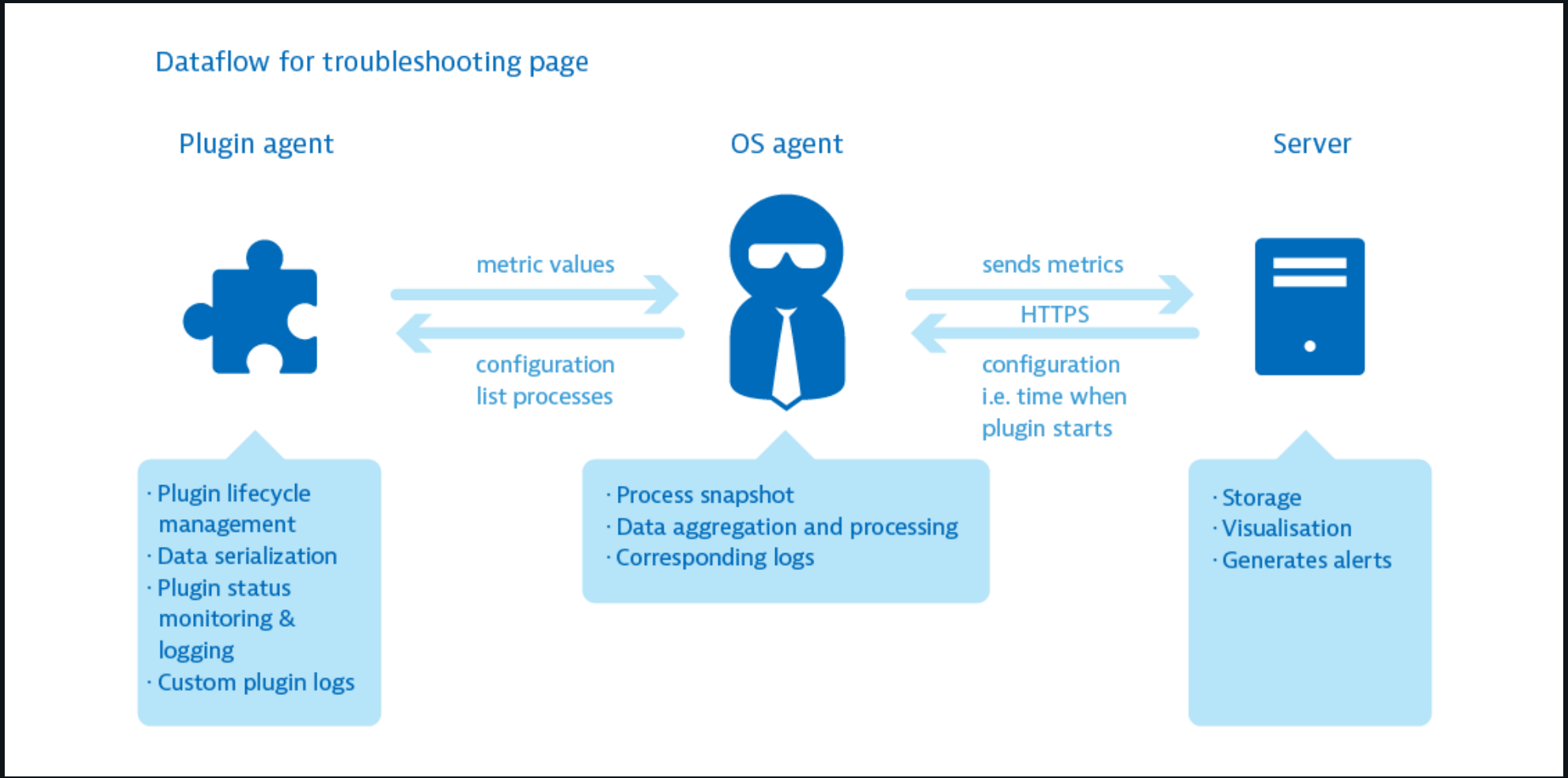
Types of plugins

- Built-in plugins
  - Dynatrace plugin
  - JMX monitoring
  - PMI monitoring
  - Infrastructure insights
  - Service insights

- Custom plugins
  - JMX plugins
  - Python plugins



High-level architecture



Built-in Plugins

- Dynatrace plugin
  - Turn on/off monitoring of detected technologies
  - Input additional environment configuration (database credentials, etc.)
- JMX monitoring
- PMI monitoring
- Infrastructure insights
  - Log analytics and Network traffic monitoring
- Service insights
  - Code level visibility on Java, .Net, etc.

Plenty of built-in plugins

NoSQL

Couchbase
Cassandra
CouchDB
Elastic
Apache Solr

Cache

Redis
Memcached
HAProxy
PHP FPM

Big Data

Hadoop
Apache Spark

Queue

Rabbit MQ
ActiveMQ

App server

WebSphere
HornetQ
jetty
Apache Tomcat

SQL

MySQL
Postgre
Mongo DB
Microsoft SQL

Custom Plugins

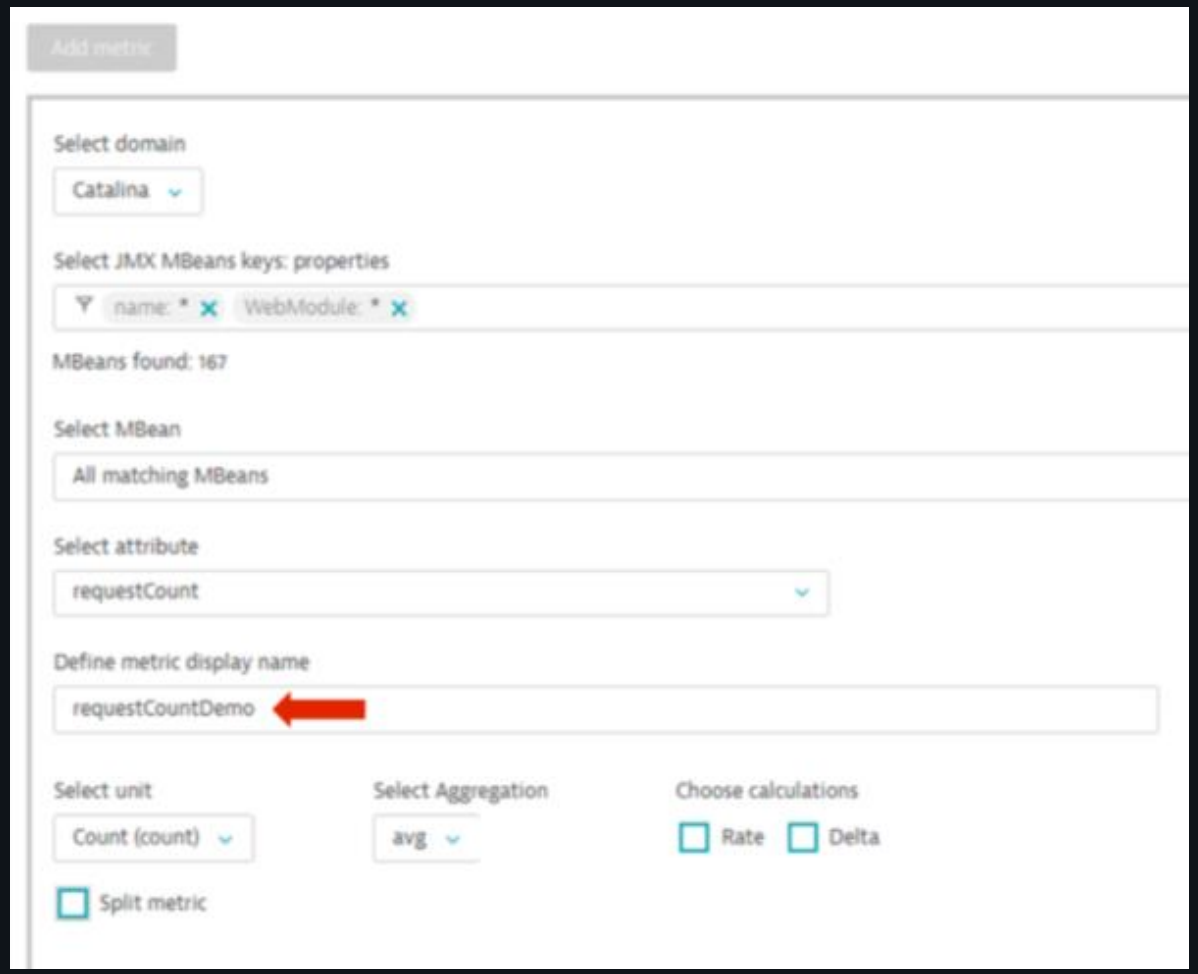
- With custom monitoring plugins you can:
  - Create and deploy new plugins that support your organization's unique monitoring needs
  - Enjoy total flexibility, decide what you need and how the results will be displayed
  - Use the power of Dynatrace AI—your custom alerts are correlated and included in root cause analysis
  - Use our Python SDK, which is equipped with advanced troubleshooting capabilities
  - Take advantage of custom metrics for processes
    - Custom metrics are displayed alongside the standard set of OneAgent performance metrics

Custom Plugins

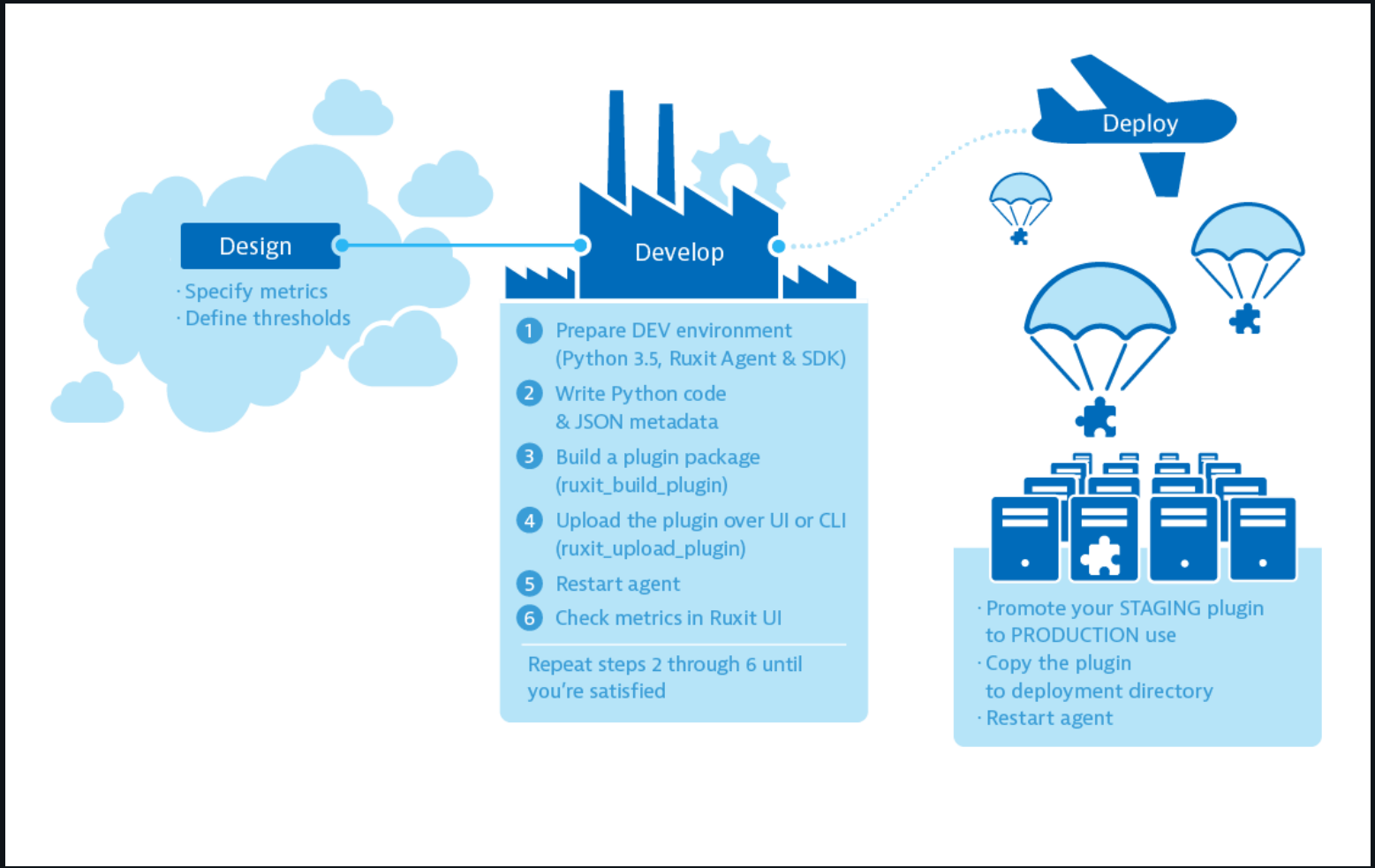
- Two types of plugins
  - JMX (JSON)
  - Python (JSON + Python script)
- Python plugins can be written for
  - hosts (e.g. additional system metrics)
  - process groups (e.g. apps, DBs, load balancers)
- SDK documentation is available on GitHub
  - https://dynatrace.github.io/plugin-sdk/index.html
  - https://help.dynatrace.com/monitor-cloud-virtualization-and-hosts/plugins/how-do-i-monitor-jmx-metrics-in-my-java-applications/
  - https://github.com/dynatrace-innovationlab/JMX-Extensions

JMX plugin editor

- Choose a domain
- Select Mbeans and the attribute
- Define your display name
- Select unit and aggregation



Local OneAgent plugin



Prerequisites

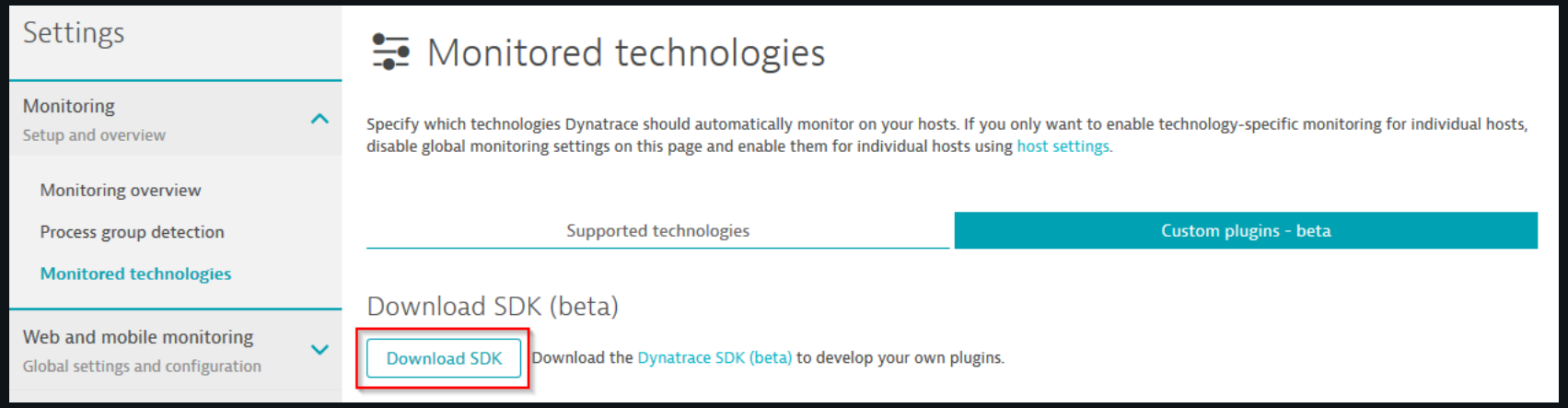
- Python 3.5/3.6
  - Download and install from http://www.python.org
  - Make sure that py (or python) works in a command prompt

```
C:\Other\plugin-sdk-1.119.186.20170606-123122>py
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ^Z
```

- OneAgent installed and running on the system
  - Download and install from your environment

Prerequisites - 2

- OneAgent SDK python package downloaded and installed
  - Download the OneAgent SDK from your environment



  - Update pip and setuptools in case they are dated
    - py -m pip install -U pip setuptools
  - From within the unzipped OneAgent SDK directory, install the python modules and scripts
    - py -m pip install oneagent_sdk-1.119.186.20170606.123122-py3-none-any.whl

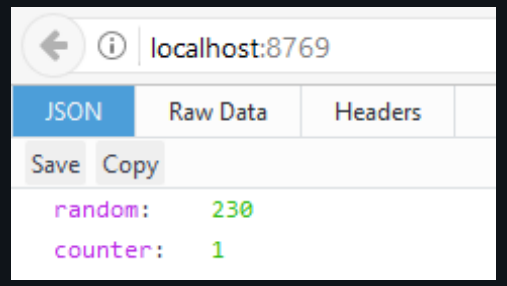
SDK demo plugin

Demo application

- The OneAgent SDK comes with a demo application
  - Launch the module using py –m plugin_sdk.demo_app

```
C:\Other\plugin-sdk-1.119.186.20170606-123122\examples\demo_plugin>py -m oneagent_sdk.demo_app
Bottle v0.12.13 server starting up (using WSGIRefServer())...
Listening on http://localhost:8769/
Hit Ctrl-C to quit.
```

  - Listens on port 8769 and returns a json



  - "random" is a random number and "counter" increases by one each time the URL is called
  - For additional parameters, such as authentication and changing the port, use python -m plugin_sdk.demo_app --help

Demo plugin - Python code

- Found at [OneAgent SDK]/examples/demo_plugin/demo_plugin.py
  - Combination of Python and SDK modules
  - The query method executes once per minute
  - Metric results - relative / absolute / per second / event
  - The plugin must be linked to an entity
- Process snapshot – [OneAgent]/log/plugin/pluginDevLoggerOsAgentDefault.log
  - A plugin needs to have executed once before the file is available

```
import requests
import json
import logging
from ruxit.api.base_plugin import BasePlugin
from ruxit.api.snapshot import pgi_name


class DemoPlugin(BasePlugin):
    def query(self, **kwargs):
        pgi = self.find_single_process_group(pgi_name('oneagent_sdk.demo_app'))
        pgi_id = pgi.group_instance_id
        stats_url = "http://localhost:8769"
        stats = json.loads(requests.get(stats_url).content.decode())
        self.results_builder.absolute(key='random', value=stats['random'], entity_id=pgi_id)
        self.results_builder.relative(key='counter', value=stats['counter'], entity_id=pgi_id)
```

Demo plugin - JSON

- Found at [OneAgent SDK]/examples/demo_plugin/plugin.json
  - name needs to be unique
  - version needs to be updated every time the JSON is changed
  - type is always python for custom plugins
  - entity defines if it should run when it finds a specific process group (recommended) or continuously
  - processTypeNames is based on the process type list in: https://dynatrace.github.io/plugin-sdk/api/plugin_json_apidoc.html
  - package and className tells the OneAgent where it can find the main plugin class
  - install_requires lists dependencies
  - activation: Singleton states that there only should be one instance
  - metrics lists the data which will be gathered by the plugin

```
{
  "name": "custom.python.demo_plugin",
  "version": "1.1",
  "type": "python",
  "entity": "PROCESS_GROUP_INSTANCE",
  "processTypeNames": ["PYTHON"],
  "source": {
    "package": "demo_plugin",
    "className": "DemoPlugin",
    "install_requires": ["requests>=2.6.0"],
    "activation": "Singleton"
  },
  "metrics": [
    {
      "timeseries": {
        "key": "random",
        "unit": "Count"
      }
    },
    {
      "timeseries": {
        "key": "counter",
        "unit": "Count"
      }
    }
  ]
}
```

Additional JSON metadata

JSON properties - metrics

- Metric source to pass parameters to the python code

```
"source": {
        "query": "db_stats['objects']"
}
```

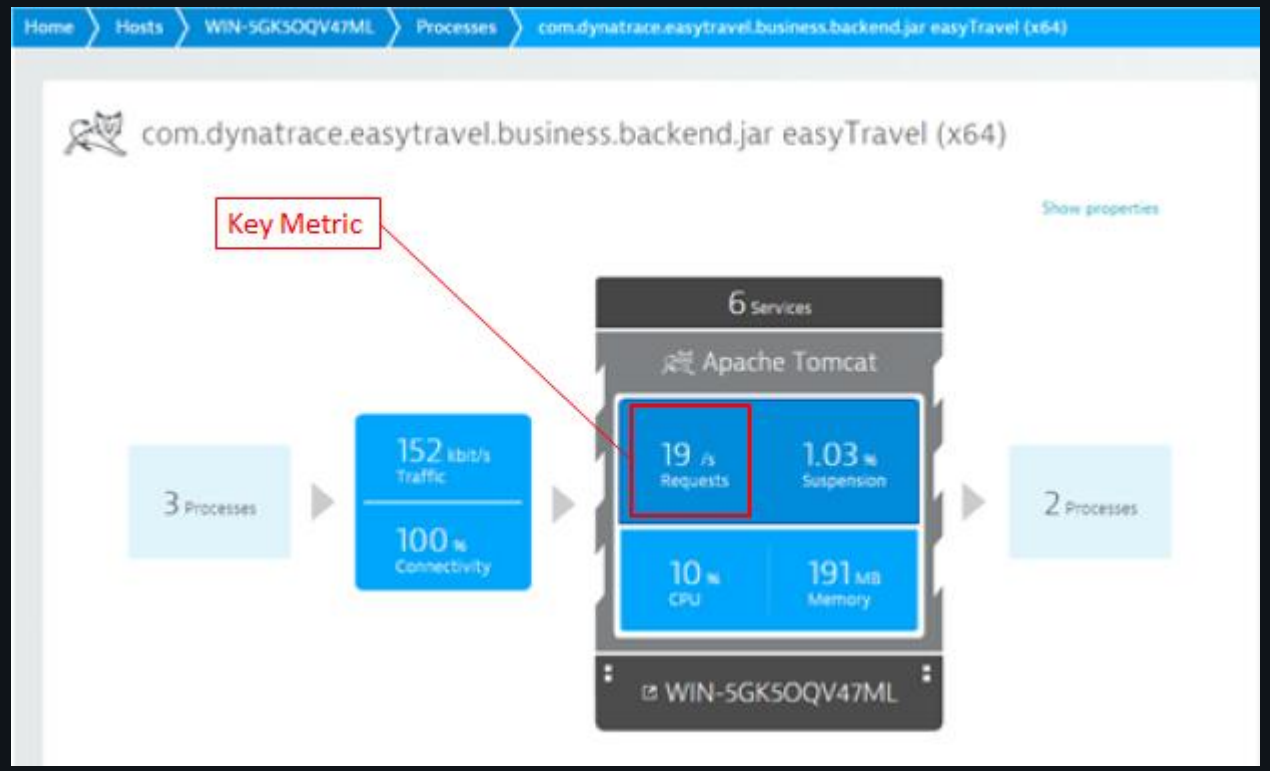
- Metric alert_settings
  - Can be set on each metric to define custom thresholds

```
"alert_id": "counter_alert_high",
"event_type": "PGI_CUSTOM_PERFORMANCE",
"event_name": "Enormous counter rate",
"threshold": 10.0,
"alert_condition": "ABOVE",
"samples": 5,
"violating_samples": 3,
"dealerting_samples": 5
```

JSON properties - UI

- keymetrics can be used to display up to two metrics on the infographic

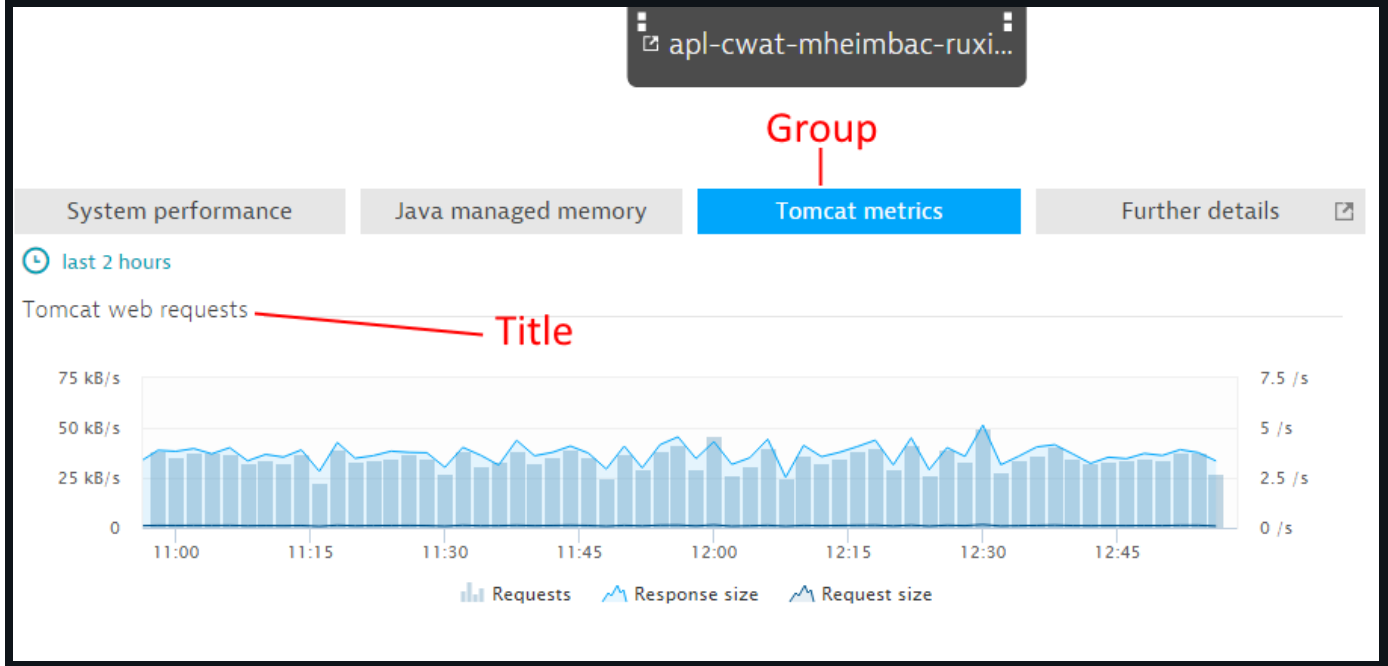
```
"ui" :
{
        "keymetrics" : [
                {
                "key" : "requestCount",
                "aggregation" : "avg",
                "mergeaggregation" : "sum",
                "displayname" : "Requests"
                }
        ],
        "keycharts" : [ ],
        "charts": [ ]
}
```



JSON properties - UI - 2

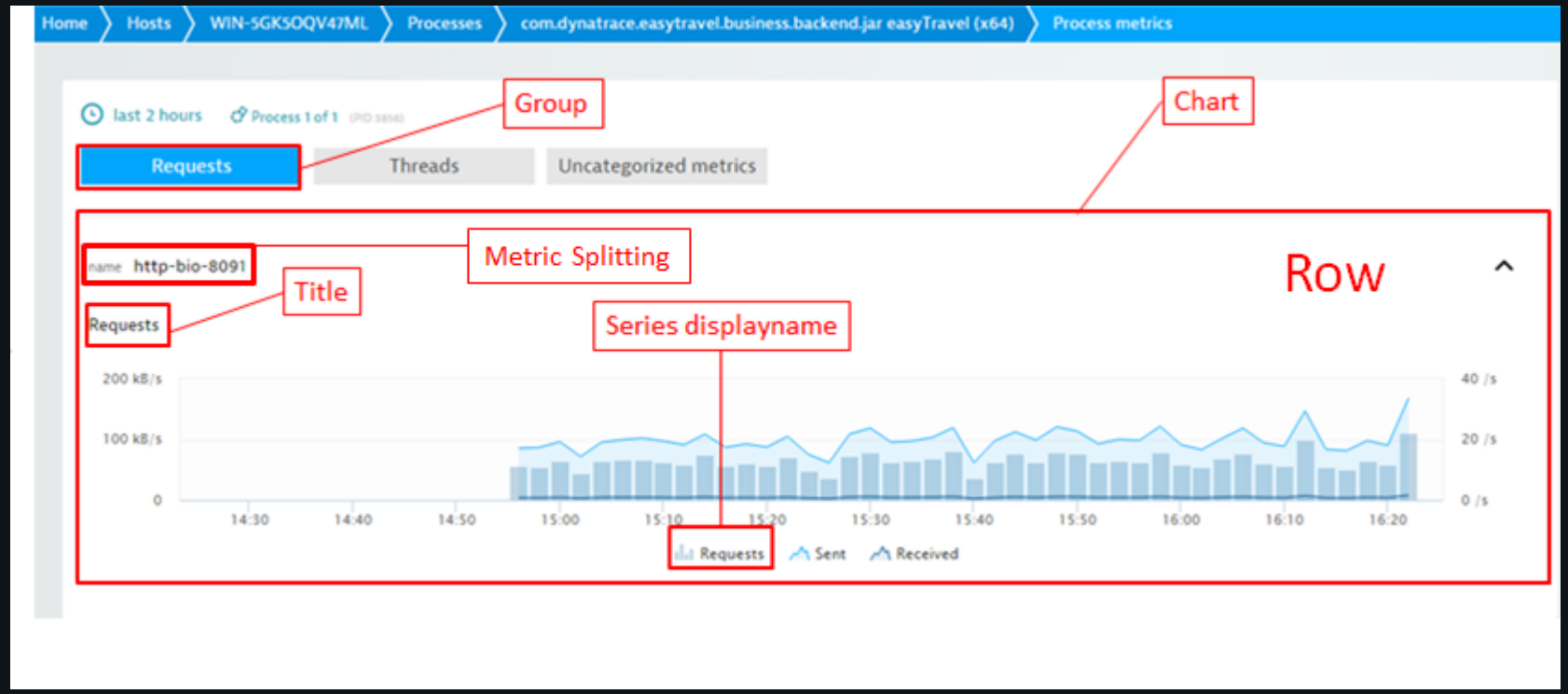
- keycharts can be used to add additional metrics directly on the process screen

```
{
    "group": "Section Name",
    "title": "Chart Name",
    "series": [
     {
            "key": "MetricName",
            "aggregation": "avg",
            "displayname": "Display name for metric",
            "seriestype": "area"
     },
     {
            "key": "Other Metric Name",
            "aggregation": "avg",
            "displayname": "Display name for metric",
            "color": "rgba(42, 182, 244, 0.6)",
            "seriestype": "area"
     }
    ]
}
```



JSON properties - UI - 3

- charts can be used to customize how the metric is charted



JSON properties - config

- configUI can be used to provide configuration to the plugin from within Dynatrace with additional properties such as displayOrder and displayName

```
"configUI" :{
        "displayName": "HAProxy",
        "properties" : [
                { "key" : "url", "displayName": "URL", "displayOrder": 3, "displayHint": "http://localhost:8080/haproxy-statistics" },
                { "key" : "auth_user", "displayName": "User", "displayOrder": 1 },
                { "key" : "auth_password", "displayName": "Password", "displayOrder": 2 }
        ]
 }
```

- If configUI configuration is required, the configUI property keys also needs to be added within the properties array

```
"properties" : [
        {
                "key" : "url",
                "type" :  "String",
                "defaultValue" : "https://localhost/haproxy_stats_ssl"
        },
        {
                "key" : "auth_user",
                "type" :  "String",
        },
        {
                "key" : "auth_password",
                "type" :  "Password",
                "defaultValue" : "password"
        }
 ]
```

Plugin simulator

Plugin simulator overview

- Mimics the OneAgent to run plugins without running a OneAgent or connecting to a Dynatrace server
- Limitations
  - The process snapshot must be provided manually
  - Plugin configuration (if any) needs to be provided manually
  - No data is sent to or received from Dynatrace Server – cannot test JSON UI properties
  - The Python environment must contains the libraries required by the plugin

Plugin simulator JSON

- Demo version found at [OneAgent SDK]/examples/demo_plugin/simulator_snapshot.json

```
{
  "entries": [
    {
      "process_type": 29,
      "group_name": "oneagent_sdk.demo_app",
      "processes": [
        {
          "process_name": "python3.5",
          "properties": {
            "CmdLine": "-m oneagent_sdk.demo_app",
            "WorkDir": "/home/demo",
            "ListeningPorts": "8769"
          }
        }
      ]
    }
  ]
}
```

- process_type can be found at https://dynatrace.github.io/plugin-sdk/api/plugin_json_apidoc.html
- The simulator runs from the directory of the simulator_snapshot.json and can be started with oneagent_simulate_plugin or oneagent_sim
  - If the program cannot be found, make sure you add [pythondir]\Scripts to your path
- If the plugin requires parameters a json file can be linked to using -r

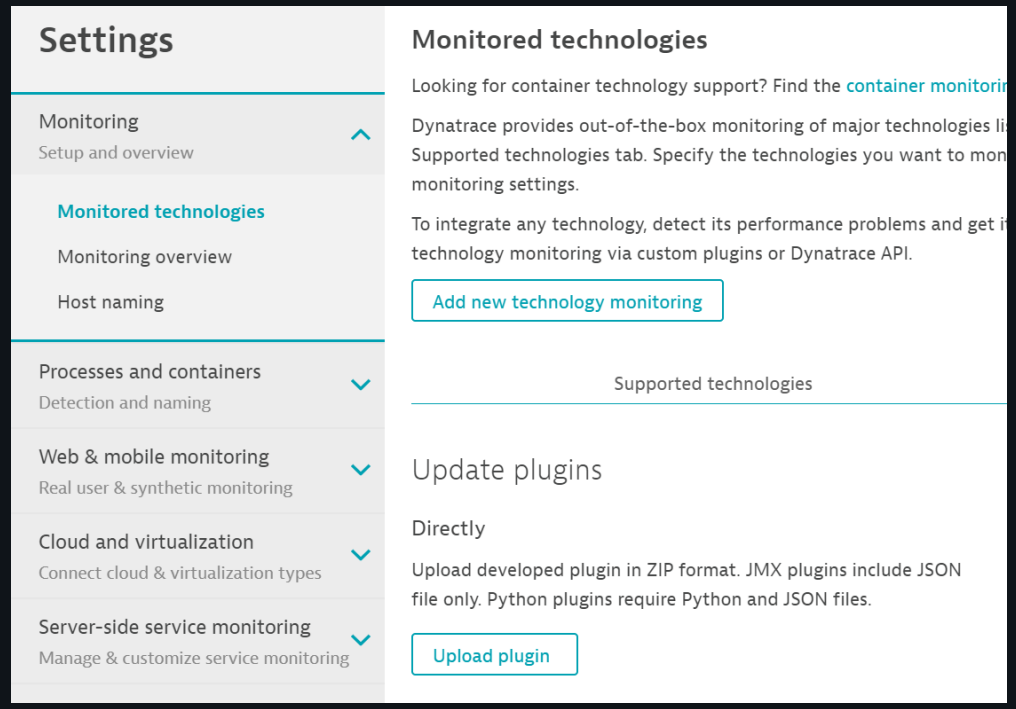
```
{
  "user": "ruxit",
  "password": "ruxit"
}
```

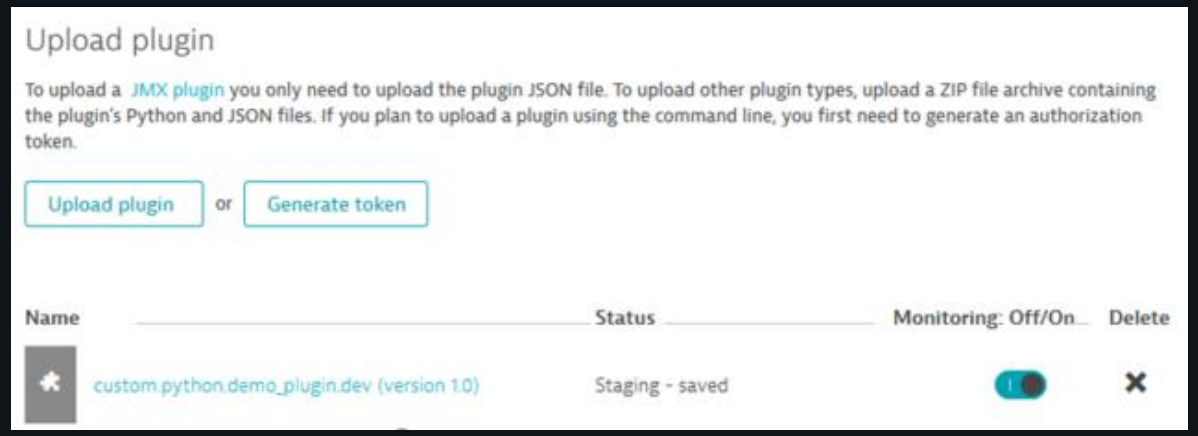
Package and deploy a plugin

Packaging the plugin

- Plugins can either be built and uploaded in separate steps, or in a single command
- To package the plugin into a zip, run oneagent_build_plugin --no_upload
  - The plugin will be deployed [oneagent path]/plugin_development/[plugin name]
  - The plugin will also be zipped into [oneagent path]/plugin_development/[plugin name].zip, ready for upload into Dynatrace
  - Make sure that you have access rights on the folder

Deploying the plugin manually

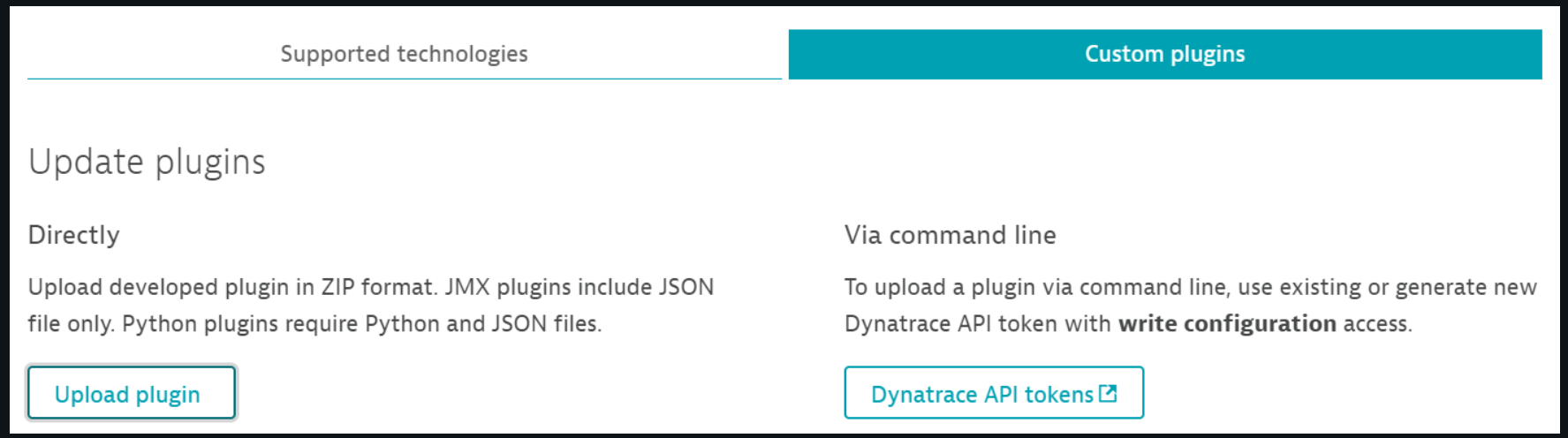




- Once the plugin is added to the OneAgent, the OneAgent needs to be restarted to be able to use the new plugin, unless it was already done during the build of the plugin

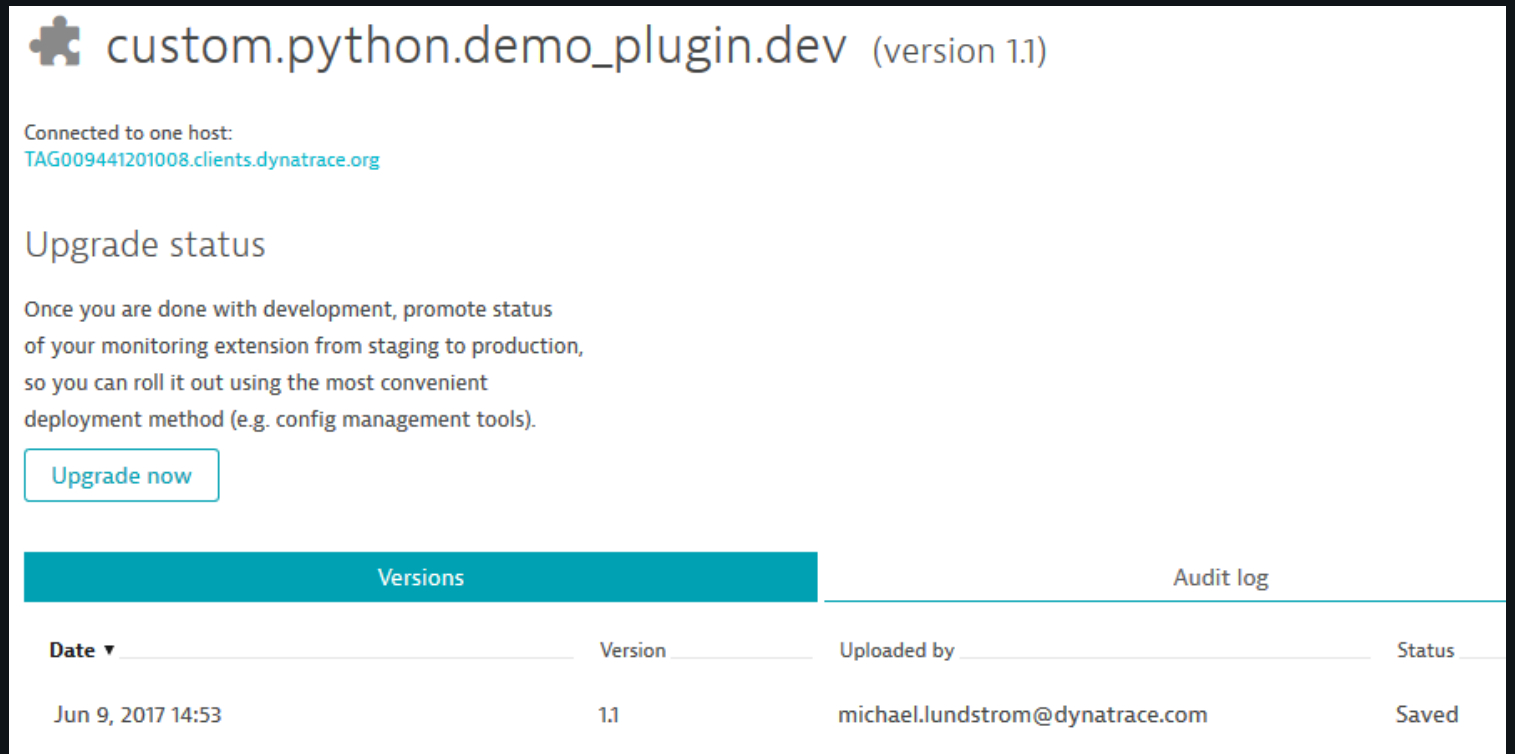
Deploying the plugin automatically

- To deploy the plugin using a command requires a token
  - Tokens expire after 30 days
- The token can be stored in a local file, an environment variable or passed directly into the upload tool
  For example oneagent_upload_plugin -t pEnLDvdMTA2lhJ_IUNC8U
- You can also build, upload and restart using a single command, oneagent_build_plugin



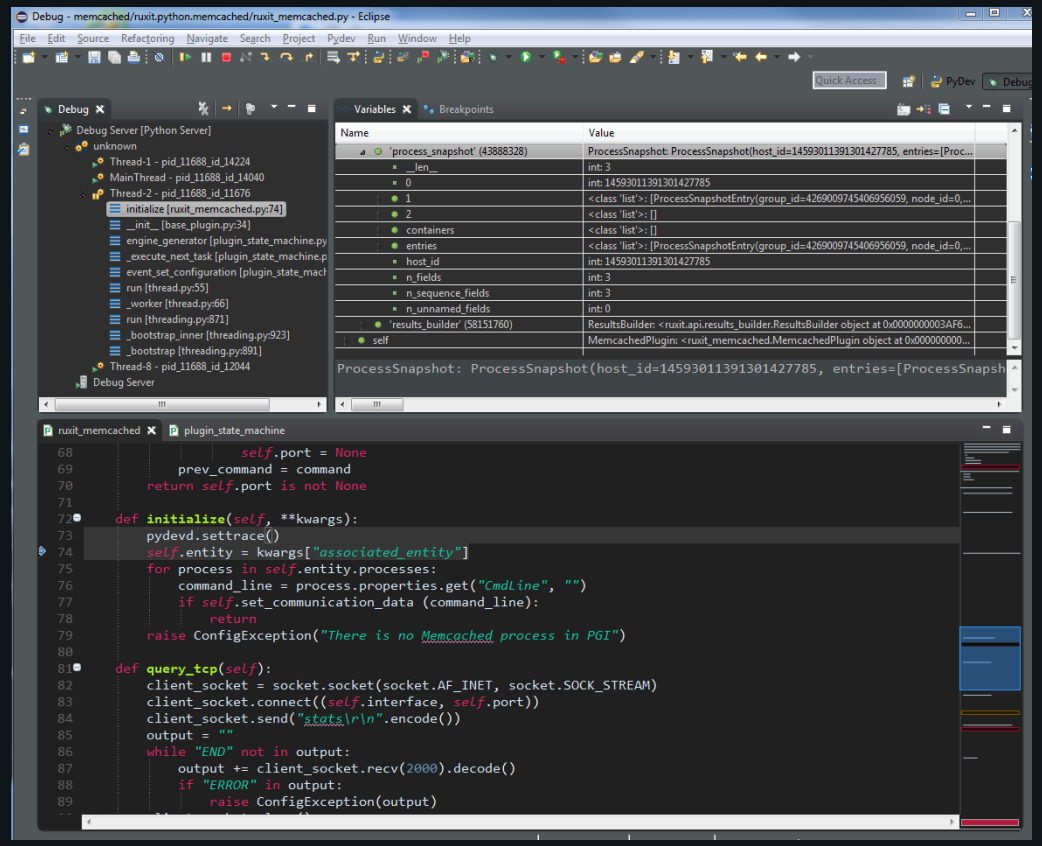
Promoting the plugin to production

- By clicking on the plugin in the interface one can choose to upgrade the plugin from staging to production. This will allow for rolling out the plugin to different hosts
  - Make sure to move the plugin from the development to deployment folder on your host
  - If the plugin needs to run on other hosts as well, copy the plugin to the deployment folders and restart the agent



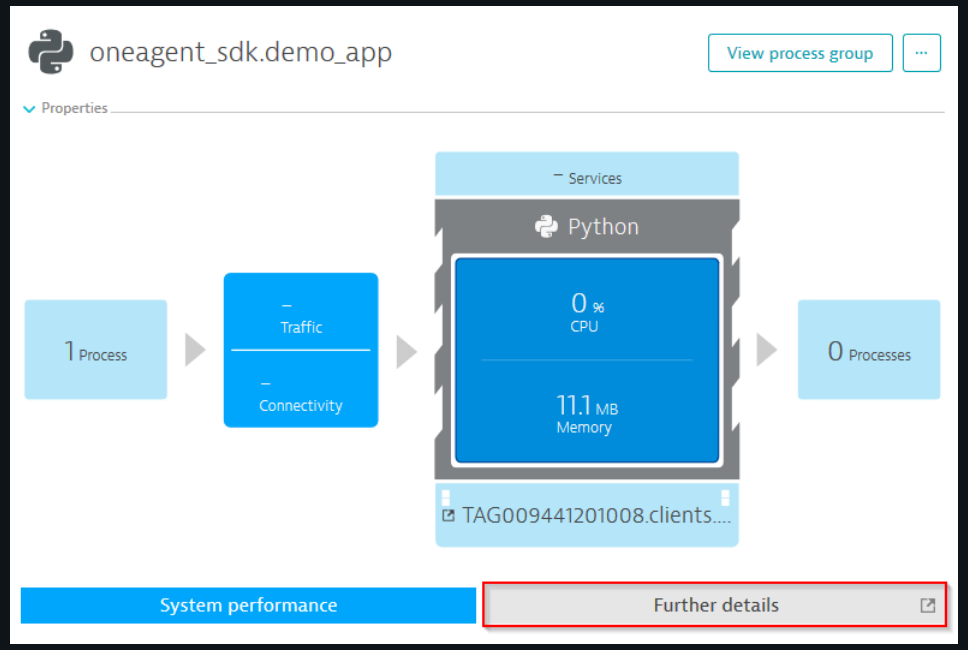
Debugging

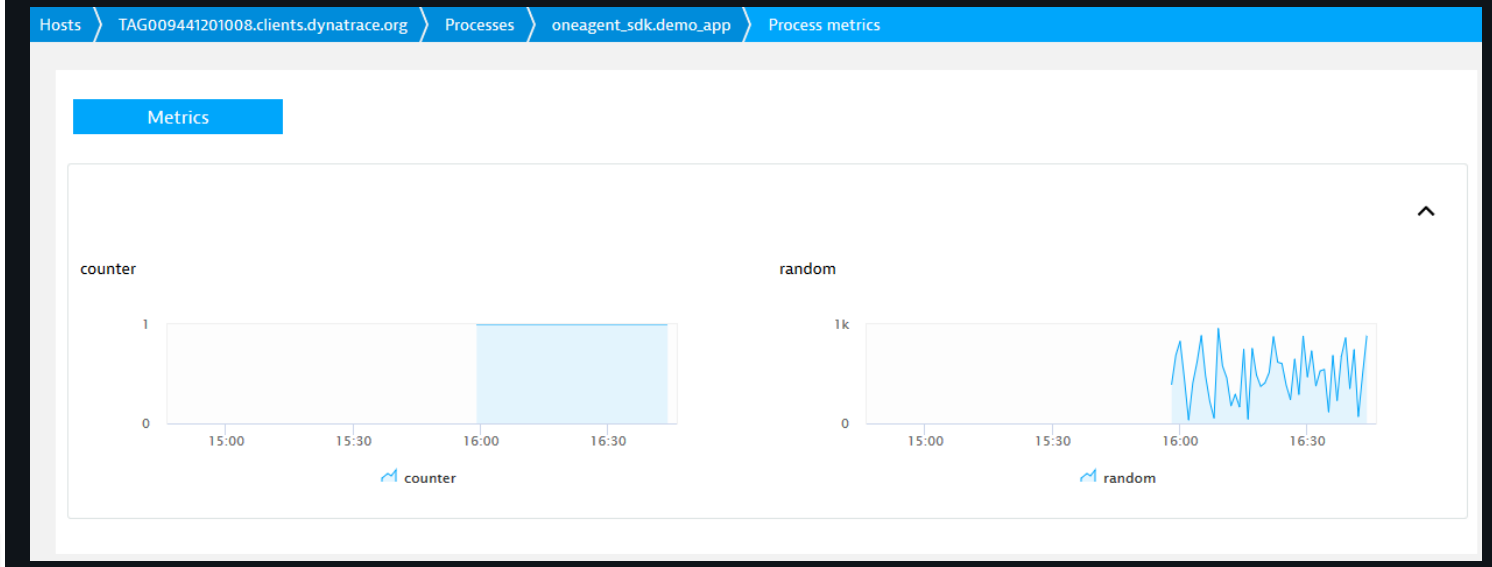
- There are several troubleshooting pages on the plugin-sdk git. One very handy possibility when creating a complicated plugin is how to debug the plugin in eclipse using breakpoints and stepping: https://dynatrace.github.io/plugin-sdk/troubleshooting/debugging.html



Visualize non-key metrics

- The metrics captured by the plugin appear in the process under “Further details”





Extending plugins

Use properties instead of fixed values

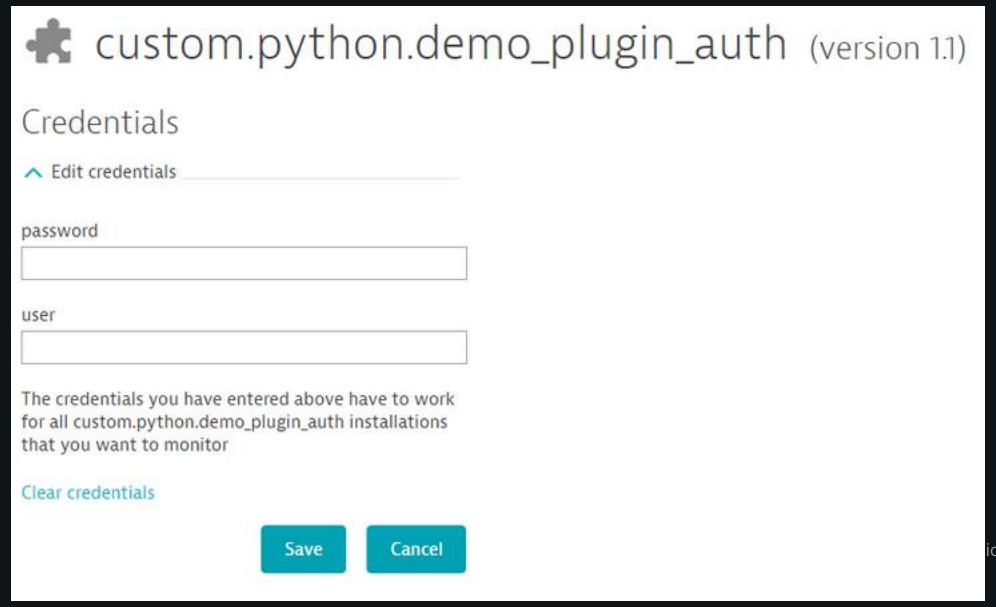
- By defining custom properties in the JSON information such as passwords do not have to be stored in clear text within the plugin

```
"properties": [
  {
    "key": "user",
    "type": "String"
  },
  {
    "key": "password",
    "type": "Password"
  }
]
```

- The properties can be referenced in the python code

```
class DemoPlugin(BasePlugin):
    def query(self, **kwargs):
        config = kwargs["config"]
        user = config["user"]
        password = config["password"]
```

- The properties can be edited from within Dynatrace

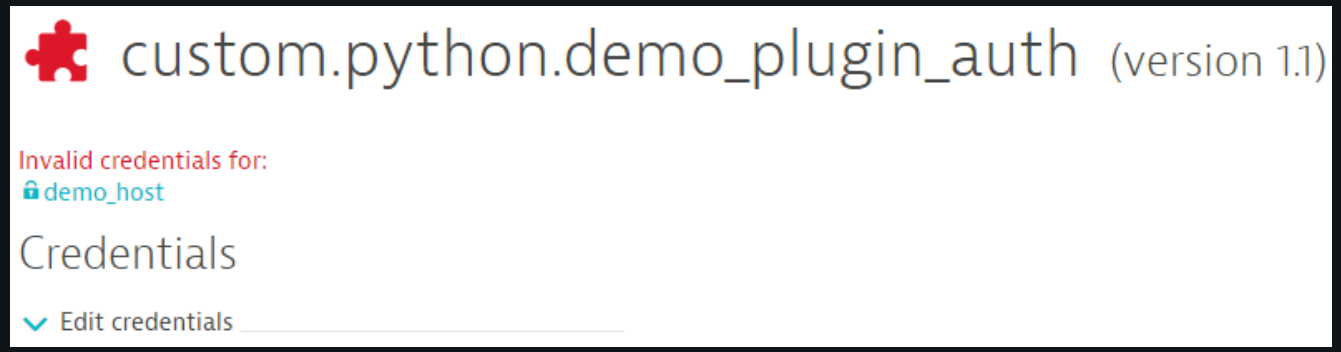


Error handling

- By capturing exceptions and passing them to the OneAgent it is possible to signal issues from within the plugin

```
try:
    response = requests.get(stats_url, auth=(user, password))
    if response.status_code == 401:
        raise AuthException(response)
    stats = json.loads(response.content.decode())
except requests.exceptions.ConnectTimeout as ex:
    raise ConfigException('Timeout on connecting with "%s"' % stats_url) from ex
except requests.exceptions.RequestException as ex:
    raise ConfigException('Unable to connect to "%s"' % stats_url) from ex
except json.JSONDecodeError as ex:
    raise ConfigException('Server response from %s is not json' % stats_url) from ex
```

- There are three exceptions, ConfigException, AuthException and NothingToReportException
- If an error occurs it is visualized on the plugin screen



Monitor several applications with one plugin

- There is no limitation within a plugin in regards to how many applications can be monitored with one plugin.
- If you, for example, have two applications running on different ports and you want to monitor both of them, you can use find_all_process_groups instead of find_single_process_group and then iterate over the result

```
_APPL_PREFIX='oneagent_sdk.demo_app'

def query(self, **kwargs):
    '''
    Scan process snapshot for groups with name starting with given prefix, and
    for all matching this prefix find ports on which they are listening. Then use this port
    for building URL to query with http. The result is json with two variables which are
    put into results with entity id of given process group.
    '''
    config = kwargs["config"]
    user = config["user"]
    password = config["password"]
    # search process snapshot using criteria defined by lambda expression
    pgi_list = self.find_all_process_groups( lambda entry: entry.group_name.startswith(self._APPL_PREFIX))
    for pgi in pgi_list:
        pgi_id = pgi.group_instance_id
        self.logger.info( "Demo pgid=%x application=%s" % (pgi_id,pgi.group_name ))
        port = None
        for process in pgi.processes:
           port = process.properties.get("ListeningPorts", None)
           break
        if port is None:
            raise ValueError( "no port definition for process group with id=%d" % pgi_id )
        # build URL for quering
        url = "http://127.0.0.1:" + port
```

Remote Extensions

Remote Extensions

- Dynatrace Extensions allow an Environment ActiveGate to reach out and remotely collect metrics from 3rd party products
- These extensions will enable Dynatrace the ability to monitor the health and performance of their instances
- Dynatrace developed Extension Examples:
  - Apigee Edge
  - Citrix NetScaler
  - Citrix virtual Apps and Virtual Desktops
  - IBM iSeries (AS/400)
  - IBM MQ
  - Juniper Networks
  - SAP ABAP platform
  - IBM DataPower
  - F5 BIG-IP LTM
  - Windows Server

ActiveGate Extensions

- Build your own ActiveGate Plugins
  - Deployment instructions as well as a simple example plugin are available

- https://www.dynatrace.com/support/help/extend-dynatrace/activegateplugin-sdk/activegate-plugins-intro/

Build ActiveGate Plugin with Python

With ActiveGate Plugins, you can integrate into Dynatrace monitoring any remote technology exposing an interface. For example, PaaS technologies, network devices, cloud technologies, or any other where OneAgent installation is not an option. ActiveGate Plugins (aka Remote Plugins) are executed on ActiveGate and can acquire metrics and topology from remote sources.

How to add ActiveGate Plugin

1. Download ActiveGate Plugin SDK.

   Download Plugin SDK SDK version 1.179

2. Write your plugin with our ActiveGate plugin help

3. Upload your plugin here or via command line.

   If you use the UI to upload the plugin make sure that the plugin is also deployed on every ActiveGate running the plugin.

   Upload plugin

Questions?

dynatrace

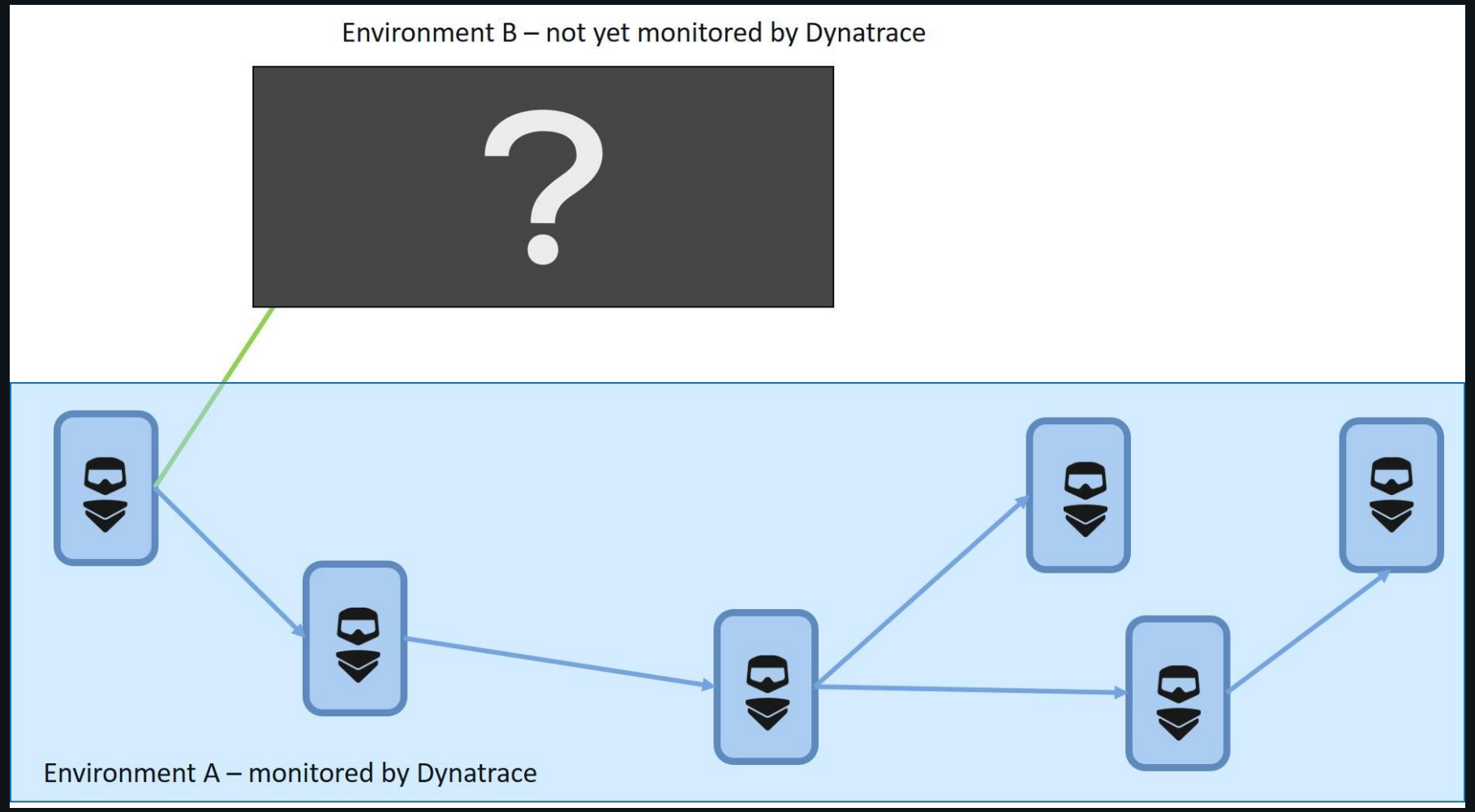
Simply smarter clouds

W3C Trace Context

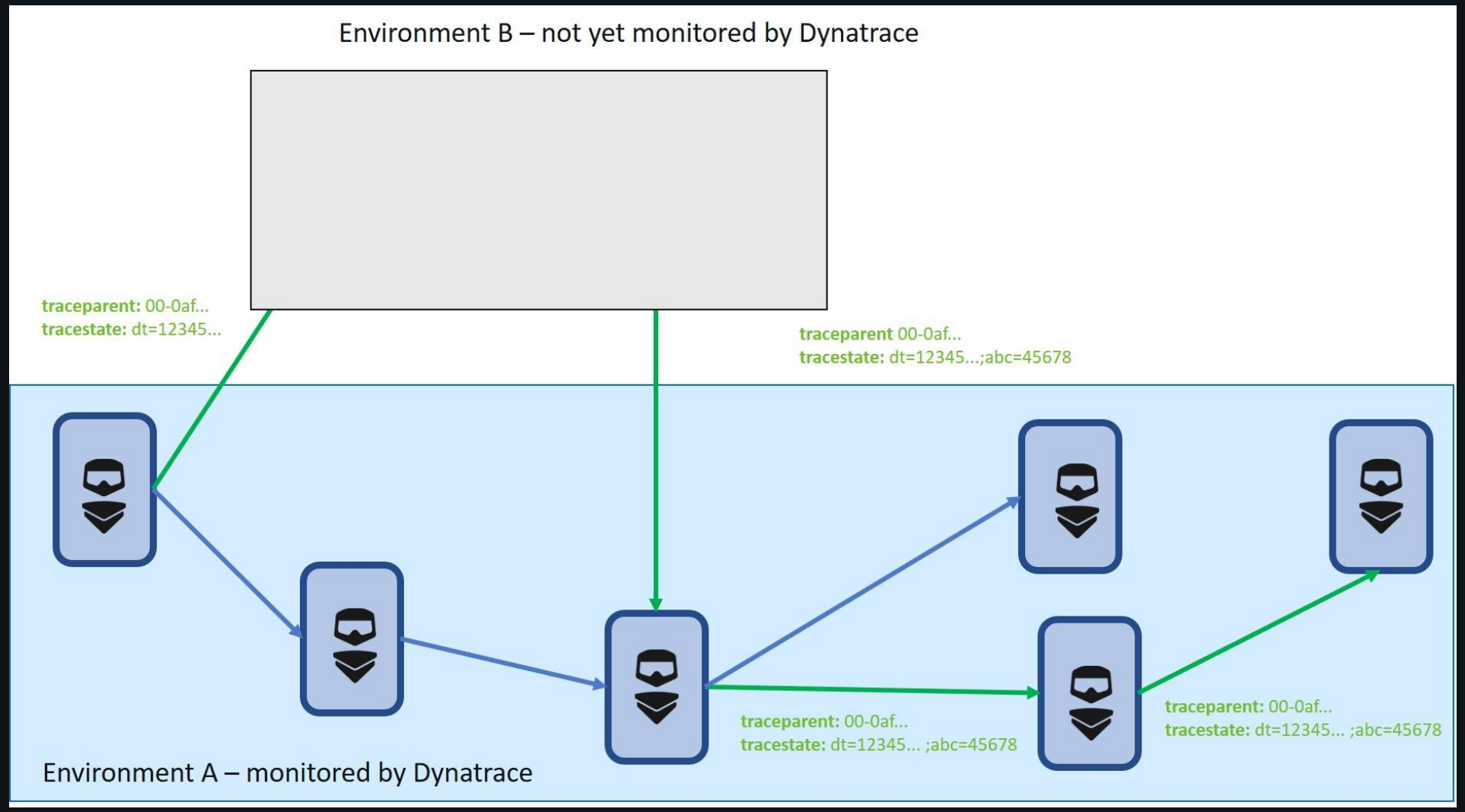
What is W3C Trace Context?

- Unified approach to context and event correlation required for distributed tracing
- Vendors often use their own defined headers to distributed tracing, a common standard would allow for forwarding of standardized headers over custom headers which are often blocked or not passed through unmonitored tiers.
- Trace Context is now a candidate recommendation from the W3C, and we expect cloud vendor services and framework developers to comply with this standard in the future.
- W3C Trace Context specification

Without W3C Trace Context - Lose Context without Dynatrace on each tier



With W3C Trace Context - Follow Context through tiers without Dynatrace



With W3C Trace Context – Search for and Identify Trace ID for tool correlation

Transactions & services > BookingService > Details > PurePaths > PurePath

getRecentBookings

Close details

Summary | Timing | Errors | Code level | Analyze

HTTP method	POST
Response status	200 - OK
Application ID	easyTravel Business Backend
Context root	/
Server name	localhost
Client IP	172.16.124.22
Web service class	com.dynatrace.easytravel.business.s
Web service name	BookingService
Web service method	getRecentBookings
Web service namespace	http://[redacted]
Web service framework	Apache Axis
Web service technology	SOAP
Thread name	http-bio-8091-exec-29
Thread ID	176
PID	6424

Client-side metadata

URI	http://[redacted]/services/BookingService/
HTTP method	POST
Response status	200 - OK
Target IP	172.16.124.23
Port	80
Web service endpoint	http://[redacted]/services/BookingService/
Web service operation	getRecentBookings

Request header

content-type	application/soap+xml; charset=UTF-8; action="urn:getRecentBookings"
host	[redacted]
Trace ID	0xe4303b06db5c451ba56167ed69789a38
Parent span ID	0x6a3c73f90184d887

Response header

content-type	application/soap+xml; action="urn:getRecentBookingsResponse";charset=UTF-8

Showing the most recent PurePath of requests of 'BookingService'

Today, 10:23 - 12:23 (2 Hours) | Apply

Filtered by Trace ID: 0xe4303b06db5c451ba5... ✕

Filter table for...

Name	Time ▾	Response time	CPU time	Method	Response code	Details
getRecentBookings /services/BookingService/	2019 May 13 11:23:20.896	166 ms	7.61 ms	POST	200	

Search Dynatrace using Trace ID