

OneAgent Ruby Agent Documentation

Spis treści

1. Lic	ensing3
2. Ins	tallation3
3. Lic	ense4
4. Op	erating4
4.1.	Configuration
4.2.	Debug logs
4.3.	Disabling agent
4.4.	Collecting names of ruby Views as Request Attributes5
4.5.	Coexistence of other APM Agent7
5. Cu:	stom metrics collection7
5.1.	Metrics collection for Kubernetes / OpenShift / Docker environment9
6. Cu:	stom instrumentation10
6.1.	Manual instrumentation 10
6.2.	Instrumentation using config file11
6.3.	Instrumentation environment variable11
6.4.	Overwriting incoming WebRequest urls to controller#action naming12
7. Sai	nple full YAML file content14
8. Sai	nple Environment Variable Config14
9. Eai	lyAccess14
9.1.	Kafka instrumentation (tested on ruby 2.5+) 15
9.2.	RabbitMQ (Bunny) instrumentation (tested on ruby 1.9+)
9.3.	Resque instrumentation (tested on ruby 1.9+) 15
9.4.	Sneakers instrumentation (tested on ruby 2.1+)15
9.1.	Delayed Jobs instrumentation (tested on ruby 2.1+)15
9.2.	Memcached instrumentation (tested on ruby 2.1+)15
9.3.	ActionView Subscriber instrumentation (tested on ruby 2.1+) 15
9.4.	Automatic Log / Trace correlation for Log Monitoring + Grail integration
9.5.	Manual Log / Trace correlation for Log Monitoring + Grail integration

1. Licensing

Our agent is licensed based on RAM consumption in similar way as Dynatrace OneAgent:

Max. RAM	License
1.6 GB	0.10
4 GB	0.25
8 GB	0.50
16 GB	1.0
32 GB	2.0
48 GB	3.0
64 GB	4.0
80 GB	5.0
96 GB	6.0
112 GB	7.0
nx16 GB	n

Single license cost is calculated on OneAgent Host Unit license and normally it's 25% of Agent Cost.

2. Installation

Supported version of ruby is 1.9 +. Right know jRuby is not supported. Ruby Agent is distributed as GEM file with native extension. This means that it must be installed in the way that will execute native code building.

First you have to add a reference to our library into your Gemfile:

gem 'onesdk_ruby'

Our gem is not available in any public repository, so you cannot just use bundle install command.

Easiest way is create directory in root of application: /your_app_dir/vendor/cache (if it's not exist) and put your gemfile there. Then you can just execute:

bundle install --local

and gem should be installed without any issues.

Alternative way: You can use local repository if available in your environment. In such case you will be able to install gemfile without copying file.

You can install as well our gem from public repo with credentials sent via e-mail:

Installation for you will need few steps (it's not in docs because it's special for you):

- You need authenticate in gemfury (our gem artifactory) bundle config gem.fury.io KEY_RECEIVED_VIA_EMAIL or export BUNDLE_GEM__FURY__IO=KEY_RECEIVED_VIA_EMAIL
- Add in you Gemfile this definition (you need to have open traffic to this url): source '<u>https://gem.fury.io/spicasolutions/</u>' do gem 'onesdk_ruby', '~> 1.0.28'
 - end
- 3. Execute bundle install
- 4. Run your application on Kubernetes cluster

3. License

License has to be puted to onesdk.yaml file

license: LICENSEKEY

or via environment variable:

ONESDK_LICENSE=LICENSEKEY

4. Operating

4.1. Configuration

Our agent is made over OneAgent SDK. In such case you don't have to create any extra configuration. You should install OneAgent in Full Stack mode to run Ruby Agent properly and restart the application. After that and detection of traffic by OneAgent you should be able to see detected ruby's services in Dynatrace.

Name		Response time median $\ensuremath{\oplus}$	Failure rate 👙	Requests \Rightarrow
0	10.132,0.6 on port: 80 ruby	102 ms	0 %	0 /min
G	10.132.0.6:80 nginx	126 ms	0 %	0 /min
0	Redis ruby	2.11 s	0 %	10 /min
0	ruby on port 3000 ruby	4.01 ms	0 %	0 /min
0	SessionsController ruby	27 µs	0 %	0 /min
0	Sidekiq Server Side Jobs ruby	749 µs	0 %	0 /min
0	StaticPagesController ruby	1.27 ms	0 %	0 /min
0	UsersController ruby	675 µs	0 %	0 /min
0	View ruby	1.35 ms	0 %	0 /min

From this moment agent will be running with your application after each restart like regular OneAgent.

Our agent is instrumenting / enables visibility:

• Incomming WebRequests

4

- Controllers with actions
- Views
- Database Calls
- Redis Calls
- External WebRequests
- Exceptions
- Sidekiq Jobs
- Rails framework
- Sinatra framework

4.2. Debug logs

By default, debug logs are disabled. During troubleshooting you can enable those logs using flag in onesdk.yaml config file:

To start collecting debug logs with our agent you have to create **onesdk.yaml** file in config directory and put there such key:

debug: true

As alternative you can use environment variable **ONESDK_DEBUG=true**. Environment variables has higher priority than config file

4.3. Disabling agent

Agent can be deactivated using environment variable ONESDK_MONITORING_DISABLED=true or via config file monitoring_disabled: true

When application starts, this configuration will prevent app instrumentation process. Agent will not perform any changes in code, SDK will not be initialized as well. After switching value to false or remove flag, application restart is needed.

4.4. Collecting names of ruby Views as Request Attributes

To see names of views on PurePaths you must create request attribute in Dynatrace.

89.4ms Response time Execution breakdown 89.4ms Processing time 99 % CPU 1.49 % Other 1	Request attributes Client IP: 10.332.0.0 Ruby 1	View: app/view		
Q. Search name, url, sql, attribute,				
/help 10.132.0.6 on port: 80	/help			😴 🗙
StaticPagesController	Summary	Timing	Code level	Errors
View	Process Operating system	ruby Linux - CentOS Linux 8 (Core) (x86)		
V 📮 full_title ApplicationHelper	Bitness Service type	64-bit Web request service		
ApplicationHelper	Service main technology	OneAgent SDK	1 51 4 20101022 125000	
generate_page_title	Called by service	10.132.0.6:80	1.5.1.4 20191023-125809)	
render_partial	User action	/help		
G render_partial	Request attributes			
 view prender,partial View 	Client IP Ruby View	10.132.0.0 app/views/static_pages/help.html.erb		

In such case you will be able to filter transactions using its value and see it in particular transactions.

To start collecting those values you have to go to Settings -> Server-side service monitoring - > Request attributes:

Request attributes

Define request attributes to enrich monitored requests with metadata. More...
Real-time updates are turned on. Please be aware that changes to method parameter
request attributes may cause temporary CPU spikes. If you want to disable real-time
updates, please head to the deep monitoring settings.

Define a new request attribute

Filter request attributes

Click on "Define a new request attribute":

Name it: Ruby View (or like you want) and add new data source:

Add new data source						
Data sources		Move up	/down	Enable	Delete	Edit
Capture SDK custom attribute 'dtarg_view	v' globally		▼			^
Rule applies to requests of the following process gro	oup, host group, technology, and process-group tag					
All process groups	~					
All host groups	~					
All service technologies	~					
Process group tags	~					
Request attribute source						
SDK custom attribute	~					
Attribute name						
dtarg_view						
Further restrict or process captured parameters (opt	iional) 🗸					
Cancel Save						

Filled it like on screen above. Attribute name should be set as: **dtarg_view.** Click on "Save on data source" and "Request attribute" as well. As result you should be able to see your newly created request attribute in Dynatrace:

Request attributes			
Define request attributes to enrich monitored requests with metadata. More			
Real-time updates are turned on. Please be aware that changes to method parameter			
request attributes may cause temporary CPU spikes. If you want to disable real-time			
updates, please head to the deep monitoring settings.			
Define a new request attribute			
Ruby			×
1 match found.			
Request attributes	Delete	Enable	Edit
Ruby View	×		

You don't need restart your process if it's already monitored with our agent.

4.5. Coexistence of other APM Agent

It's highly not recommended to use our agent on the same application with other solutions like NewRelic, Instana, DataDog, ScoutAPM etc. The pattern of the instrumentation is similar in all of mentioned agents, so there may produce some issues on your application.

5. Custom metrics collection

Our gem is able to collect some extra metrics from GC and Threads. List of available metrics is listed below:

onesdk.ruby count (GC count), onesdk.ruby_heap_allocated_pages onesdk.ruby_heap_sorted_length onesdk.ruby heap allocatable pages onesdk.ruby heap available slots onesdk.ruby_heap_live_slots onesdk.ruby heap free slots onesdk.ruby_heap_final_slots onesdk.ruby heap marked slots onesdk.ruby heap eden pages onesdk.ruby_heap_tomb_pages onesdk.ruby total allocated pages onesdk.ruby total freed pages onesdk.ruby total allocated objects onesdk.ruby_total_freed_objects onesdk.ruby malloc increase bytes onesdk.ruby_malloc_increase_bytes onesdk.ruby minor gc count onesdk.ruby_major_gc_count

onesdk.ruby_compact_count onesdk.ruby_remembered_wb_unprotected_objects onesdk.ruby_remembered_wb_unprotected_objects_limit onesdk.ruby_old_objects onesdk.ruby_old_objects_limit onesdk.ruby_oldmalloc_increase_bytes onesdk.ruby_oldmalloc_increase_bytes_limit onesdk.gc_total_time (ms) onesdk.ruby_thread_count

IMPORTANT:

To start collecting metrics you should enable OneAgent metric API according to this doc: <u>https://www.dynatrace.com/support/help/shortlink/local-api#enable-the-oneagent-metric-api</u>

To start collecting those metrics with our agent you have to create **onesdk.yaml** file in config directory and put there such keys:

metrics_ingest: true

As alternative you can setup Environent Variable **ONESDK_METRICS_INGEST=true** which has higher priority than config file. You can switch metrics ingest with setting its value to false.

Metric_ingest key true value is enabling metrics collection.

Because of Dynatrace limitations, right now metrics will not show up in further details tab on process group instance dashboard. They will be assigned to proper process group despite that. We cannot setup units for those metrics automatically so all of them are generic ones. This behavior will change in the future. To access collected metrics, you must:

Create custom chart or enter Data Explorer:

	Data explorer Query for metrics and transform results to gain desired insig	hts. Vie	w Document	ation 🖸					
	Build Code								
Α	onesdk	^	Auto 🗸	Split by	(include all)	Filter by	♥ (include all)	+	• …
	With data in the selected timeframe								
	No metrics found.	Key	y			onesc	lkruby.gc_count.gau	ıge	
	Without data in the selected timeframe	Un	it				Unspecifi	ied	🕨 Run query
	onesdkruby.gc_count.gauge	Las	st written sinc				Apr 1, 2021 02	:26	
	onesdkruby.gc_count.gauge	De	fault aggregat	ion			a	avg	
	onesdk.ruby_total_gc_time	Ag	gregations			auto, avg,	count, max, min, si	um	
ſ	onesdk.ruby_total_gc_time	Vie	ew all metric	informatio	on 🖸				ſ
	onesdk.ruby_total_freed_pages				-			di seni Manana di Angela Manana di Angela	
	onesdk.ruby_total_freed_pages								
l	onesdk.ruby_total_freed_objects								J
(onesok.ruby_total_treed_objects		men reque	U JCI VICC I	anure itu	постиссь на	тезрасез ву	Largest Kubernetes clusters i	n
			rate		CP	Il requests		terms of CPU	

✓ Average ✓ Split by Process X Host X onesdk.ruby_total_gc_time Filter by 🛛 🕆 (include all) + Add Metric Graph 🎁 4.5 1.5 0 14:52 14:56 15:04 14:54 14:58 15:00 15:02 15:06 15:08 15:10 15:12 15:14 15:16 15:18 15:20 ✓ ruby | COS-INP-SK-2

All metrics will be available on key "onesdk.metric_name".

You can split values by process group and host. Such metric can be pinned to your dashboard.

5.1. Metrics collection for Kubernetes / OpenShift / Docker environment

Because of how OneAgent is instrumentic containers, there is only part of agent that collects data from code. Rest of agent is operating on different layer. This is why we cannot use OneAgent Metric API, we need to use <u>Metrics API V2</u>. In such scenario we need to do few steps:

- 1. <u>We need to create API Token with **metrics.ingest** permission: <u>How to generate Api</u> <u>Token in Dynatrace</u></u>
- 2. We need to determine what type of connection between OneAgent in POD and Dynatrace we have. We can be connected directly to Dynatrace server. In such case we have such uri patterns for metrics ingestion:
 - a. Managed https://fyour-domain/e/fyour-environment-id/api/v2/metrics/ingest
 - b. SaaS <u>https://{your-environment-id}.live.dynatrace.com/api/v2/metrics/ingest</u>

If we are using Environment ActiveGate, it will looks like this (remember to add used port, default is 9999):

- c. Environment ActiveGate <u>https://{your-activegate-domain}/e/{your-environmentid}/api/v2/metrics/ingest</u>
- 3. We need to pass those 2 values to agent via Environment Variable or config file.
 - a. Environment Variables: export ONESDK_METRICS_API_URL=https://<DT_URL_OR_AG_URL>/api/v2/metrics/ingest export ONESDK_METRICS_API_TOKEN=<API_TOKEN> export ONESDK_METRICS_INGEST=true
 b. Config file
 - metrics_ingest: true
 metrics_api_url: https://<DT_URL_OR_AG_URL>/api/v2/metrics/ingest
 metrics_api_token: <API_TOKEN>

After that start your application and proper metrics should be available for you in metrics explorer.

6. Custom instrumentation

It is always possible to instrument extra pieces of code if initial instrumentation is not enough. It may happen if your application is using multiple custom classes that should be exposed as Custom services.

6.1. Manual instrumentation

If agent is initialized and it is working properly on your application, you can always import our class and use it to instrument your code manually. Below you can find example:

First you have to import onesdk on top of your file:

require 'onesdk'

Then inside your method initialize tracer:

tracer = Onesdk.onesdk_customservicetracer_create(Onesdk.onesdk_asciistr("Method name"),
Onesdk.onesdk_asciistr("Custom Service Name"))

Start the tracer Onesdk.onesdk_tracer_start(tracer)

Then execute your method code.

If you want to report exception you can do this like that (before end of the tracer).

```
Onesdk.onesdk_tracer_error(tracer, Onesdk.onesdk_asciistr("Exception Class Name"),
Onesdk.onesdk_asciistr("Exception message"))
```

Exception can be reported only once per tracer, if you have several exceptions you have to combined them as single string.

Before method finish add line like below:

Onesdk.onesdk_tracer_end(tracer)

If you would like to report request attributes, add a below command before finishing tracer:

```
Onesdk.onesdk_customrequestattribute_add_string(Onesdk.onesdk_asciistr("KEY"),
Onesdk.onesdk_asciistr("STRING_VALUE"));
```

To expose them in Dynatrace, you have to create proper Request Attribute configuration.

© Spica Solutions 2022

6.2. Instrumentation using config file

To use this feature you have to create yaml file in [app_dir]/config/onesdk.yaml

```
Sample content is:
custom_instrumentation: True/False (enables disables custom instrumentation)
methods:
        - class_module: string (Module::Class or Module or Class)
        method: string (method name)
Sample file content:
custom_instrumentation: True
methods:
```

- class_module: User method: send_password_reset_email
- class_module: User method: send_activation_email
- class_module: User method: activate
- class_module: User method: authenticated?
- class_module: App::ApplicationHelper method: full_title

If instrumentation go fine, you will be able to see log entries related to particular method instrumentation. You can select multiple methods from single module/class

After changing config file, you have to restart ruby application to see effect.

6.3. Instrumentation environment variable

If you prefer using environment variable for config changes, you can setup instrumentation of custom methods in the same way as via vonfig file. First you have to export environment variable that will enable this module:

export ONESDK_CUSTOM_INSTRUMENTATION=true

Second step is environment variable that will list Classes/Models and methods to be instrumented:

```
export
ONESDK_METHODS="App::ApplcationHelper|full_title\\User|send_password_reset_ema
il\\User|send_activation_email\\User|authenticated?"
```

For this environment variable syntax is

"ModuleOrClass|actionOrMethos\\AnotherModuleOrClass|anotherActionOrMethod". You can chain multiple configurations like that.

After you add environment variable, you have to restart ruby application to see effect.

6.4. Overwriting incoming WebRequest urls to controller#action naming

By default Dynatrace is naming WebRequests based on URL. This is how it works for all technologies. Ruby developers used to different approach that uses controller name and action. This is why we created Request Attribute that provides this information.

Name \$	iotai time consump- tion 🔻	meaian response time ≑	Actions
ControllerAction		1.08 s	∞ ⊽ … ∧
Top 3 values of 3			
static_pages#home		1.07 s	₩ 7
static_pages#help		1.11 s	₩ ∀ …
sessions#new		526 ms	₩ Ÿ …

To make it work you need to create definition of request attribute in Dynatrace settings:

Capture SDK custom attribute 'dta	arg_controlleraction'globally		^
Rule applies to requests of the following	process group, host group, technology, and pro	cess-group tag	
All process groups	~		
All host groups	~		
All service technologies	~		
Process group tags	~		
Request attribute source			
SDK custom attribute	~		
Attribute name			
dtarg_controlleraction			
Further restrict or process captured para	meters (optional) 🗸 🗸		

You can name this Request attribute as you like, important is to provide Attribute name filter as **dtarg_controlleraction**.

After that, new incoming web requests will be tagged. You can use request attribute for filtering and grouping transactions based on controller name and action. You can as well use it for changing original web request naming rule. To make so you need to create proper

Cancel Save

Global Web Request naming rule or one in the service. As example here is rule used in service of your choice:



Enable request naming rule

Naming pattern

Define a custom naming pattern for the requests of this service using the predefined placeholders.

{RequestAttribute:ControllerAction}	
-------------------------------------	--

Access unmasked data. Warning: This will potentially expose personalized data before it gets stored and then shown. Affected fields are marked with (*) in the placeholder section box. Do not use if you are unsure.

Condition(s)

Define the conditions that a request must meet for this request naming rule to be applied.

Request attribute ~	•	ControllerAction	~
exists 🗸			

Define if Dynatrace should access this request attribute from downstream service calls. You can optionally restrict the downstream services.



Match on downstream services

Where ControllerAction is name of created request attribute (can be different in your environment).

After that all new incoming web requests will be named based on controller#action rule (if those information's are available).

~	Oct 04 16:51:27.894	static_pages#home	364 ms	364 ms	GET	200
~	Oct 04 16:51:26.380	static_pages#help	1.11 s	1.11 s	GET	200
~	Oct 04 16:51:25.014	sessions#new	526 ms	526 ms	GET	200
~	Oct 04 16:51:23.997	static_pages#help	1.53 s	1.53 s	GET	200
~	Oct 04 16:51:22.148	static_pages#help	1.07 s	1.07 s	GET	200

Original information about url is still available in PurePath/trace.

static_pages#help				··· ×
Summary	Timing	Code level	Logs	Errors (0)
Request attributes				
ControllerAction	static_pages#help			
Metadata				
URI HTTP method Response status Application ID Context root Server name PID	/help?[declared unavailable by agent] GET 200 - OK 138.68.102.163 on port: 3000 / 138.68.102.163 204578			

7. Sample full YAML file content

```
debug: true
custom instrumentation: true
methods:
    class_module: User
   method: send_password_reset_email
    class_module: User
   method: send_activation_email
    class_module: User
   method: activate
    class_module: User
   method: authenticated?
    class_module: ApplicationHelper
    method: full_title
    class module: ApplicationHelper
   method: page_title
   class_module: ApplicationHelper
method: generate_page_title
metrics_ingest: true
pgi_id: PROCESS_GROUP_INSTANCE-9BE796BA4609E25F
```

8. Sample Environment Variable Config

```
ONESDK_DEBUG=true/false (default false)
ONESDK_METRICS_INGEST=true/false (default false)
ONESDK_CUSTOM_INSTRUMENTATION=true/false (default false)
ONESDK_METHODS="App::ApplcationHelper|full_title\\User|send_password_reset_email\\User|send_activation_email\
\User|authenticated?" (default empty)
```

9. EarlyAccess

Early access features are not tested well but should be working fine. By default, those features are disabled. In future releases this behavior will be changed to enable by default.

9.1. Kafka instrumentation (tested on ruby 2.5+)

Kafka instrumentation can be turned on by setting up environment variable **ONESDK_KAFKA_INSTRUMENTATION=true** or in config yaml by setting **kafka_instrumentation: true**

9.2. RabbitMQ (Bunny) instrumentation (tested on ruby 1.9+)

RabbitMQ (bunny) instrumentation can be turned on by setting up environment variable **ONESDK_RABBITMQ_INSTRUMENTATION=true** or in config yaml by setting **bunny_instrumentation: true**

9.3. Resque instrumentation (tested on ruby 1.9+)

Resque instrumentation can be turned on by setting up environment variable ONESDK_RESQUE_INSTRUMENTATION=true or in config yaml by setting resque_instrumentation: true

9.4. Sneakers instrumentation (tested on ruby 2.1+)

Sneakers instrumentation can be turned on by setting up environment variable **ONESDK_SNEAKERS_INSTRUMENTATION=true** or in config yaml by setting **sneakers_instrumentation: true**

Sneakers instrumentation provide automatic injection of tracing into worker classes. Using rake and Rails is mandatory to make instrumentation work.

9.1. Delayed Jobs instrumentation (tested on ruby 2.1+)

Delayed Jobs instrumentation can be turned on by setting up environment variable ONESDK_DELAYED_JOB_INSTRUMENTATION=true or in config yaml by setting delayed_job_instrumentation: true

9.2. Memcached instrumentation (tested on ruby 2.1+)

Memcached (Dalli) instrumentation can be turned on by setting up environment variable ONESDK_MEMCACHED_INSTRUMENTATION=true or in config yaml by setting memcached_instrumentation: true

9.3. ActionView Subscriber instrumentation (tested on ruby 2.1+)

ActionView Subscriber instrumentation can be turned on by setting up environment variable ONESDK_ACTIONVIEW_INSTRUMENTATION=true or in config yaml by setting actionview_instrumentation: true

Enabling ActionView subscriber will disable original view instrumentation.

9.4. Automatic Log / Trace correlation for Log Monitoring + Grail integration

Thanks to update of OneAgent SDK for C our agent now has implement of correlation between traces and logs entries. It works right now for default logger and Logger Formatter. To make it work you need enable log monitoring and log collection for your ruby process. You should use <u>Dynatrace Documentation</u> for this task.

After that you need enable environment variable:

ONESDK_LOGGER_INSTRUMENTATION=true

After that, logs collected in context of PurePath should be visible in your trace log tab.

If you don't want to connect all collected logs to trace, you can define max log severity that will be picked for that. To make it work you need to add another environment variable:

ONESDK_MAX_LOG_SEVERITY=<SEVERITY_LEVEL>

Available log severities: DEBUG, INFO, WARN, FATAL, ERROR, UNKNOWN.

If you set for example FATAL severity, all logs that are maximum on that level will be connected to trace.

9.5. Manual Log / Trace correlation for Log Monitoring + Grail integration

If you are using custom LogFormatter, for example for JSON logging, you still can have advantages of trace / log correlation but it will need slightly changes into formatter code.

```
require "onesdk"
```

end

```
#We need to collect spanid and traceid from tracer
spanid = Onesdk.onesdk_tracecontext_get_current_spanid()
traceid = Onesdk.onesdk_tracecontext_get_current_traceid()
```

```
<here you need add pgi_id, trace_id and span_id to your log according to Dynatrace documentation>
```

end

Documentation that presents how pgi_id,trace_id and span_id ingestion should looks like is available here: <u>Link to documentation</u>