

Full Stack Self-Service Diagnostics with Dynatrace



Stefan Hofbauer
Team Lead Sales
Engineer



Sergio Hinojosa
Sales Engineering
Manager



Omar Mazzeo
Sales Engineer

Prerequisites

- Personal AWS Account:
 - <https://aws.amazon.com/free/>
- Postman API Client
 - <https://www.getpostman.com/downloads/>
- Download/Clone the Repository:
 - <https://github.com/stehofbauer/Dynatrace-Hotday-2019-SelfServiceDiagnostics>
or
 - <https://tinyurl.com/yydxjm4m>

Agenda

- Formalities
 - Introductions
 - What we're covering
 - Level of Expertise required
 - Individual EC2 Instances
 - EasyTravel – Book Exciting travel to far away destinations
- Setup and Launching our EC2 instance with EasyTravel
- Configuration API
- Use Case 1 – CPU Increase
- Use Case 2 – Booking Failure
- Use Case 3 – Slow Login
- Use Case 4 – Slow Loading Recommendations
- Tips and Tricks

What we're covering/not covering

- This is an exploration of common
- This is not a product tutorial
- This is not a test
- There is no need to be an AWS expert and everything we are covering can be applied to ANY environment (e.g. Azure, GCP, Kubernetes, On-Prem, etc...)
- The goal of this sessions is:
 - Help you become more comfortable using Dynatrace to diagnose application problems
 - Make the diagnosis easy for others in your organization
 - Not everybody needs to be a "Dynatrace Expert"

How we're covering

- We will walk through several Use Case Problem patterns
- Each Student will have their own private EC2 instance running a version of EasyTravel (created via CloudFormation)
- Each Student will have their own private Dynatrace SaaS tenant
 - Tenant ID and Credentials on your attached card
- During the course of this session we will be doing the following:
 - Creating our environment via CloudFormation
 - Performing Dynatrace configurations via the UI and the API
 - Triggering EasyTravel Problem Patterns via the EasyTravel Config UI

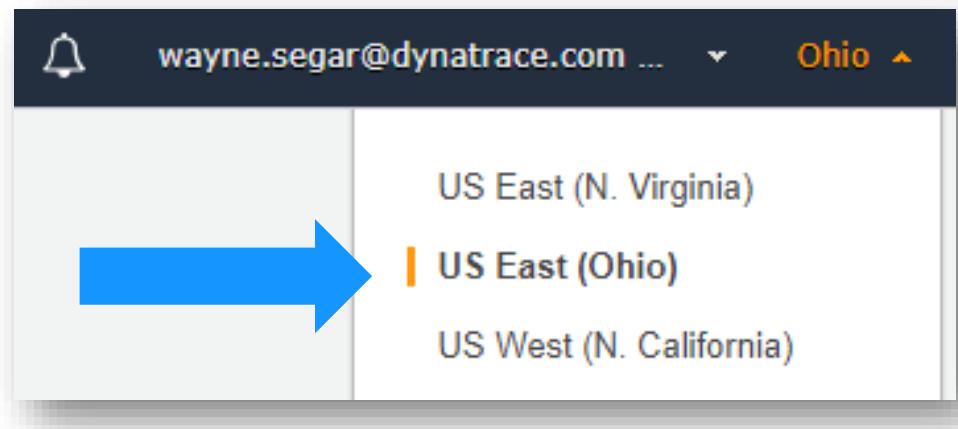
Level of Expertise

- Basic Dynatrace familiarity
- Expert certification not required
- REST API basics

Setup Work

Setup AWS: Default Region

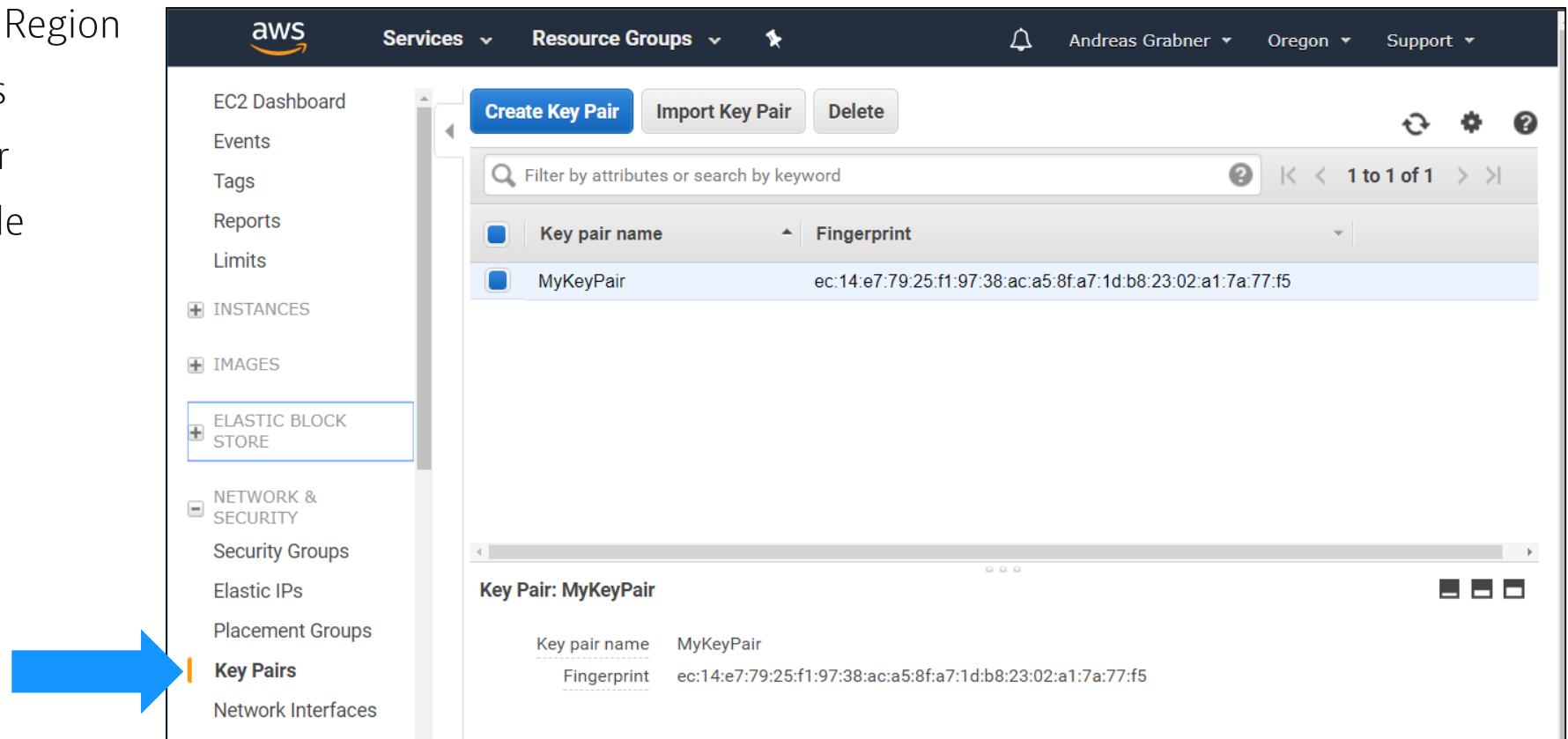
- Login to your AWS Account: <https://aws.amazon.com>
 - Validate and make note of your default region



Setup AWS: EC2 Key Pair

1. Create EC2 Key Pair

1. Go to EC2 – Pick Region
2. Click on Key Pairs
3. Create a new Pair
4. Store the .pem file

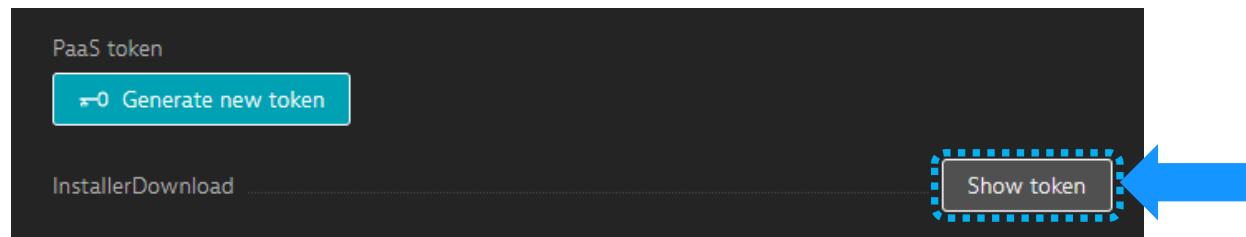


Setup Dynatrace: Retrieve API Token

- Login to your Dynatrace SaaS Tenant provided to you:

https://<TENANT_ID>.sprint.dynatracelabs.com

- Navigate to:
 - Click "Deploy Dynatrace" from the left side menu
 - Click "Start Installation"
 - Click "Set up PaaS monitoring"
 - Click "Show token"



- Copy your token to your clipboard

Launching our EC2 Instance

Our CloudFormation Template will create a Production Linux server with EasyTravel installed/running and deploy a OneAgent pointing to your Dynatrace SaaS Tenant

Launching the Stack ◀

Upload Template

Filling in the Details

Create and Confirm

Follow Events

View Outputs

Validate Environment

Cleanup

Launching the CloudFormation Stack

- What is CloudFormation?
 - A declarative way to define AWS Resources, e.g: EC2, CodeDeploy, CodePipeline, Lambda, Roles, ELBs, ...
 - Learn more about CloudFormation:
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide>
- Our Steps
 - In your AWS Console browse to the CloudFormation Service

AWS services

Find services
You can enter names, keyword or acronyms.

X

CloudFormation
Create and Manage Resources with Templates
Recently visited services

Launching the Stack

Upload Template ◀

Filling in the Details

Create and Confirm

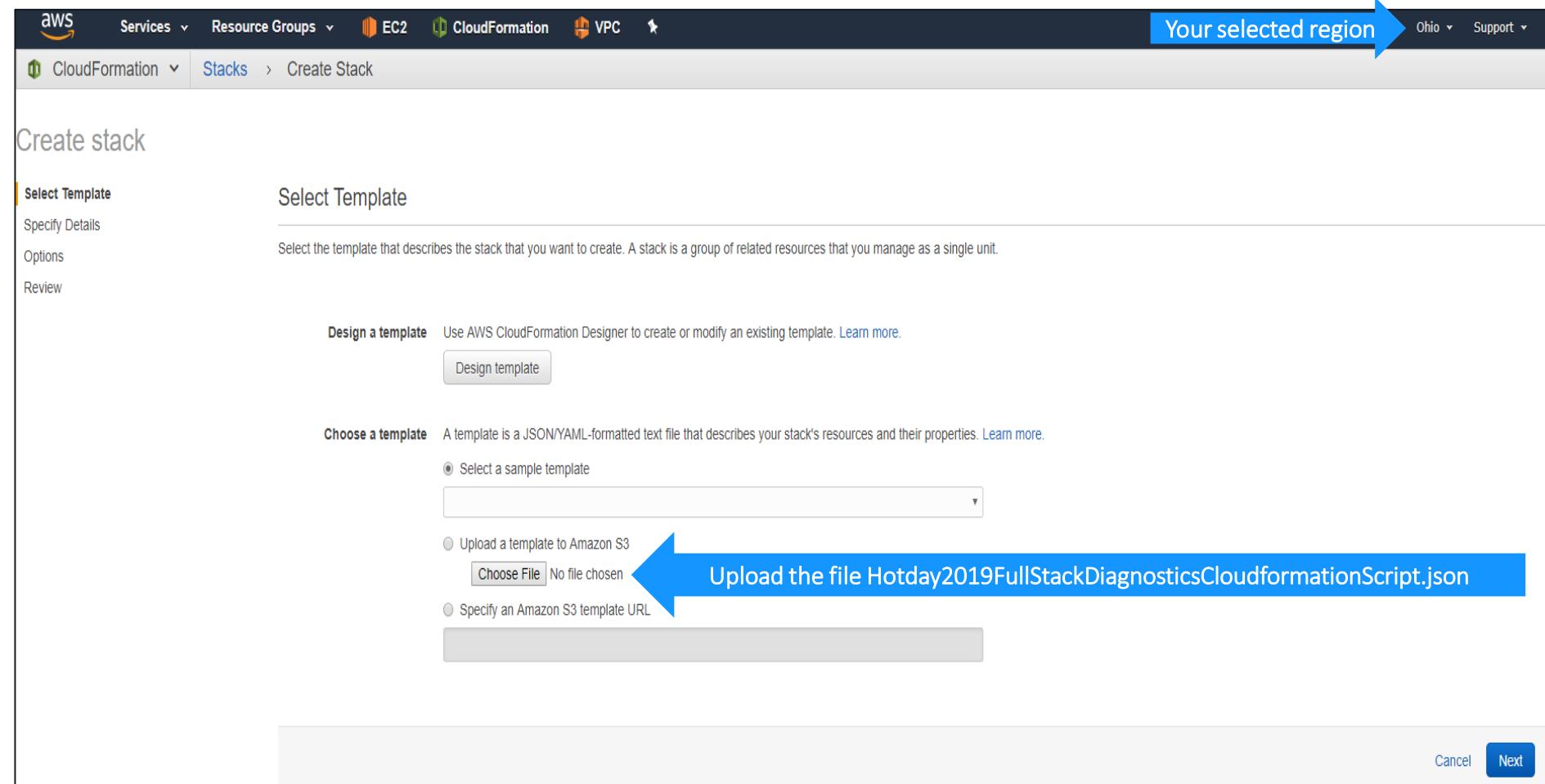
Follow Events

View Outputs

Validate Environment

Cleanup

Upload Template



The screenshot shows the AWS CloudFormation 'Create stack' wizard, Step 1: Select Template. The 'Upload a template to Amazon S3' section is highlighted with a blue arrow pointing to the 'Choose File' button. A blue box at the bottom right contains the text 'Upload the file Hotday2019FullStackDiagnosticsCloudformationScript.json'.

Services ▾ Resource Groups ▾ EC2 CloudFormation VPC Your selected region Ohio ▾ Support ▾

CloudFormation ▾ Stacks > Create Stack

Create stack

Select Template

Specify Details

Options

Review

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template Use AWS CloudFormation Designer to create or modify an existing template. [Learn more.](#)

Design template

Choose a template A template is a JSON/YAML-formatted text file that describes your stack's resources and their properties. [Learn more.](#)

Select a sample template

Upload a template to Amazon S3

Choose File No file chosen

Specify an Amazon S3 template URL

Cancel Next

CloudFormation can be found in the GitHub repository

Launching the Stack

Upload Template

Filling in the Details ◀

Create and Confirm

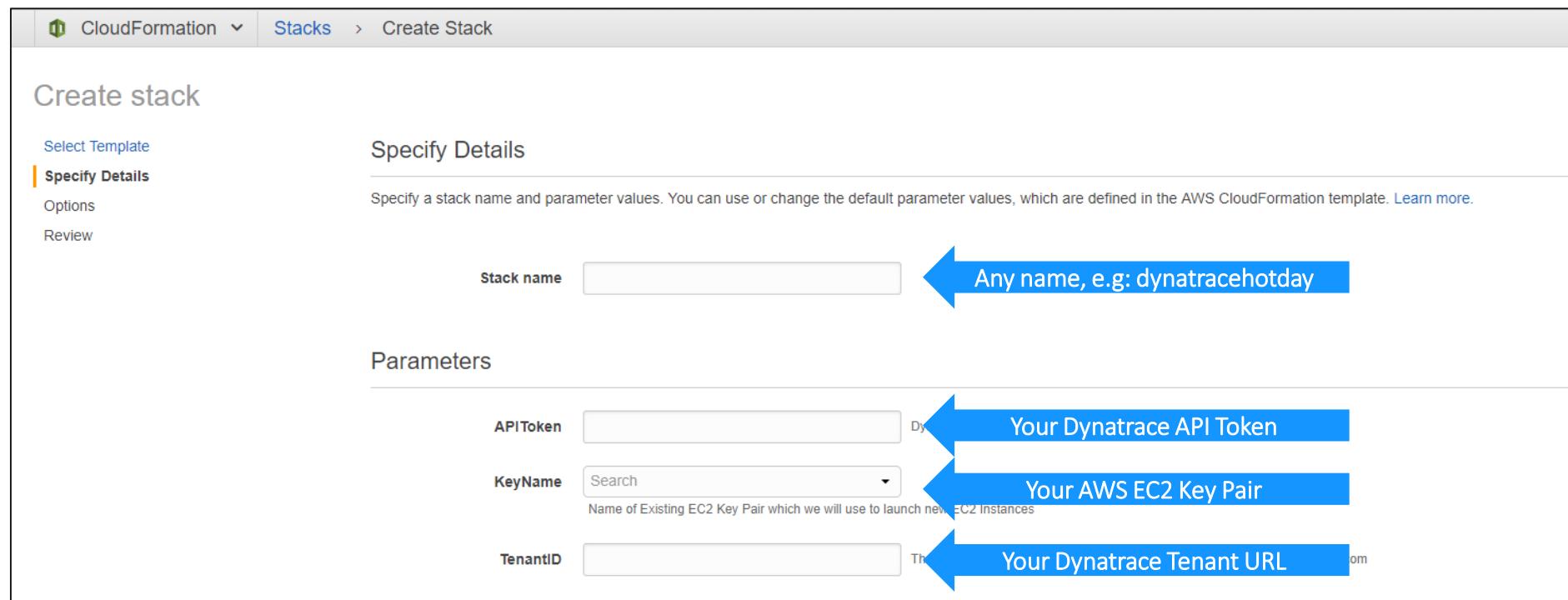
Follow Events

View Outputs

Validate Environment

Cleanup

Filling in the Details



The screenshot shows the AWS CloudFormation 'Create stack' interface. In the top navigation bar, 'CloudFormation' is selected, followed by 'Stacks > Create Stack'. The main section is titled 'Create stack' and 'Specify Details'. A sub-instruction says: 'Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)' On the left, a sidebar lists 'Select Template', 'Specify Details' (which is highlighted in orange), 'Options', and 'Review'. The 'Specify Details' section contains three input fields: 'Stack name' (with a placeholder 'My first stack'), 'API Token' (with a placeholder 'Dyantrace'), 'KeyName' (with a dropdown menu showing 'Search' and a note 'Name of Existing EC2 Key Pair which we will use to launch new EC2 Instances'), and 'TenantID' (with a placeholder 'The URL of your Dynatrace tenant'). Blue arrows point from the right side of the slide to each of these fields with their respective descriptions: 'Any name, e.g: dynatracehotday' for the Stack name, 'Your Dynatrace API Token' for the API Token, 'Your AWS EC2 Key Pair' for the KeyName, and 'Your Dynatrace Tenant URL' for the TenantID.

Launching the Stack

Upload Template

Filling in the Details

Create and Confirm ◀

Follow Events

View Outputs

Validate Environment

Cleanup

Create and Confirm

Review

Template

Template URL	https://s3.us-east-2.amazonaws.com/cf-templates-19yzke3lzb8f-us-east-2/2018365GfD-Wayne_Perform2019.json
Description	Dynatrace Perform 2019: Full Stack Self-Service Diagnostics with Dynatrace
Estimate cost	Cost

Details

Stack name: test

APIToken asdfas
 KeyName WayneSegarKeyPair
 TenantID asdfasf

Options

Tags

No tags provided

Rollback Triggers

No monitoring time provided

No rollback triggers provided

Advanced

Notification	
Termination Protection	Disabled
Timeout	none
Rollback on failure	Yes

[Quick Create Stack](#) (Create stacks similar to this one, with most details auto-populated)

[Cancel](#) [Previous](#) [Create](#)

Launching the Stack

Upload Template

Filling in the Details

Create and Confirm

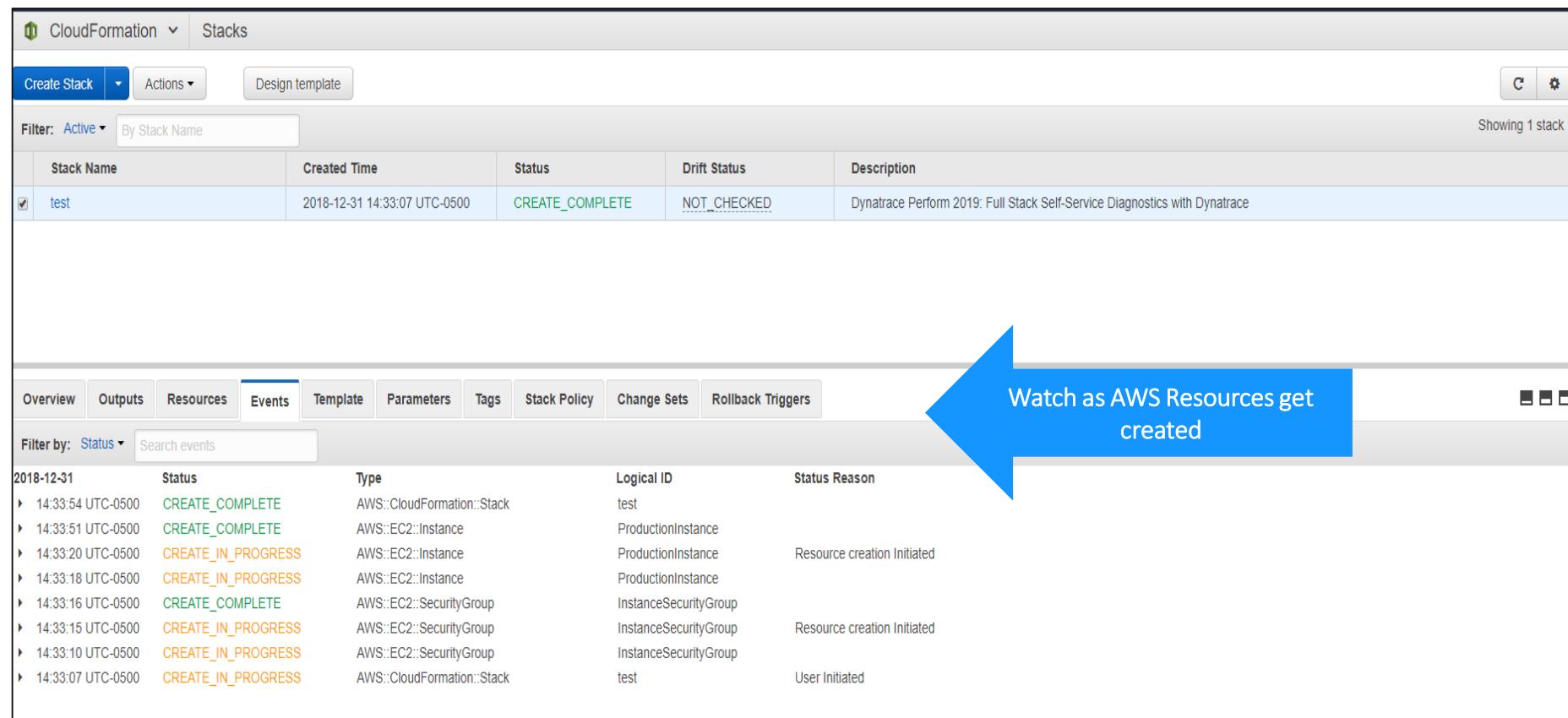
Follow Events ◀

View Outputs

Validate Environment

Cleanup

Follow Events



The screenshot shows the AWS CloudFormation console with the 'Follow Events' view selected. The top navigation bar includes 'CloudFormation' and 'Stacks' tabs, along with 'Create Stack', 'Actions', and 'Design template' buttons. A filter dropdown set to 'Active' and a search bar for 'By Stack Name' are also present. The main table displays one stack named 'test' with the following details:

Stack Name	Created Time	Status	Drift Status	Description
test	2018-12-31 14:33:07 UTC-0500	CREATE_COMPLETE	NOT_CHECKED	Dynatrace Perform 2019: Full Stack Self-Service Diagnostics with Dynatrace

Below this, the 'Events' tab is selected in the navigation bar, showing a detailed log of events:

Date	Status	Type	Logical ID	Status Reason
2018-12-31	CREATE_COMPLETE	AWS::CloudFormation::Stack	test	
14:33:54 UTC-0500	CREATE_COMPLETE	AWS::EC2::Instance	ProductionInstance	
14:33:51 UTC-0500	CREATE_COMPLETE	AWS::EC2::Instance	ProductionInstance	Resource creation initiated
14:33:20 UTC-0500	CREATE_IN_PROGRESS	AWS::EC2::Instance	ProductionInstance	
14:33:18 UTC-0500	CREATE_IN_PROGRESS	AWS::EC2::Instance	ProductionInstance	
14:33:16 UTC-0500	CREATE_COMPLETE	AWS::EC2::SecurityGroup	InstanceSecurityGroup	
14:33:15 UTC-0500	CREATE_IN_PROGRESS	AWS::EC2::SecurityGroup	InstanceSecurityGroup	Resource creation initiated
14:33:10 UTC-0500	CREATE_IN_PROGRESS	AWS::EC2::SecurityGroup	InstanceSecurityGroup	
14:33:07 UTC-0500	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	test	User initiated

Watch as AWS Resources get created

Launching the Stack

Upload Template

Filling in the Details

Create and Confirm

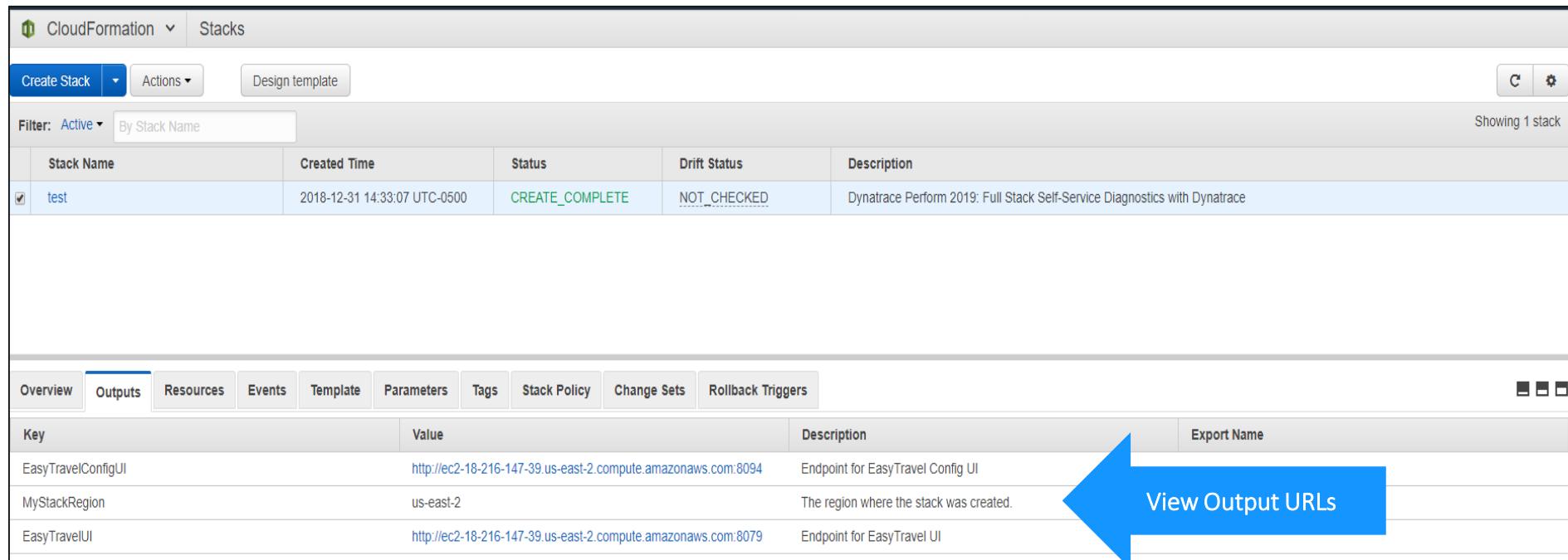
Follow Events

View Outputs ◀

Validate Environment

Cleanup

View Outputs



The screenshot shows the AWS CloudFormation 'Outputs' tab for a stack named 'test'. The table lists three outputs:

Key	Value	Description	Export Name
EasyTravelConfigUI	http://ec2-18-216-147-39.us-east-2.compute.amazonaws.com:8094	Endpoint for EasyTravel Config UI	
MyStackRegion	us-east-2	The region where the stack was created.	
EasyTravelUI	http://ec2-18-216-147-39.us-east-2.compute.amazonaws.com:8079	Endpoint for EasyTravel UI	

View Output URLs

Note: It will take roughly 10 minutes for EasyTravel to fully come up

Launching the Stack

Upload Template

Filling in the Details

Create and Confirm

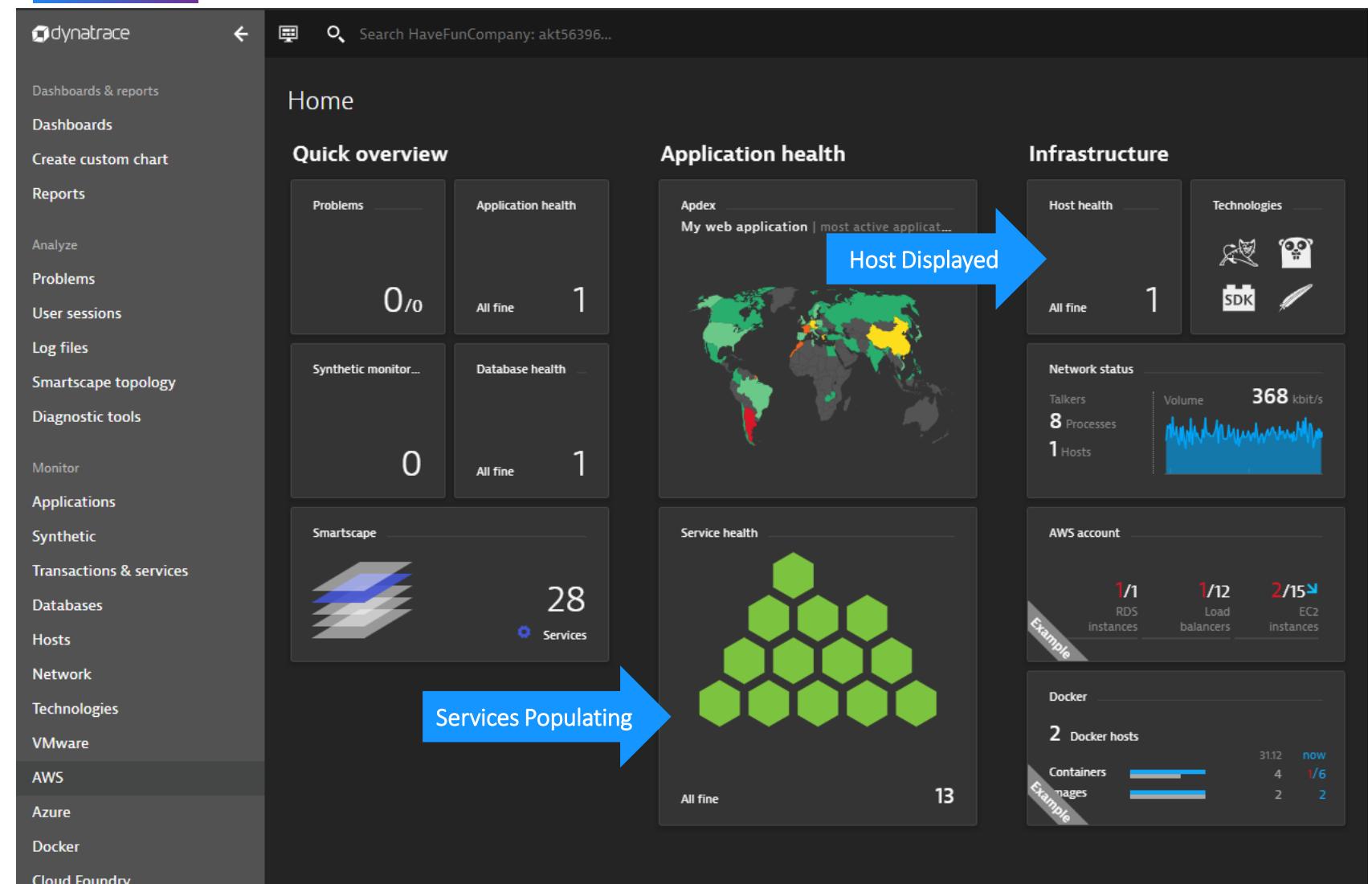
Follow Events

View Outputs

Validate Environment ◀

Cleanup

Validate Environment



Launching the Stack

Upload Template

Filling in the Details

Create and Confirm

Follow Events

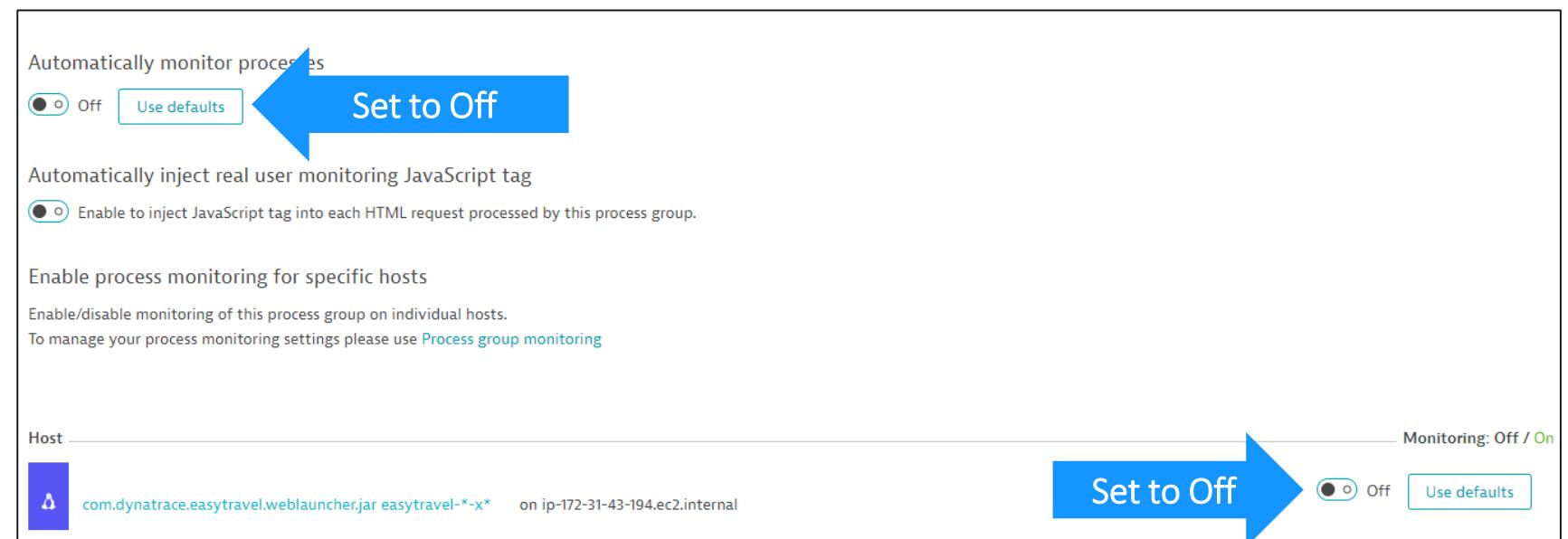
View Outputs

Validate Environment

Cleanup ◀

Cleanup

- In order to get RUM data to stitch to backend services we need to disable deep monitoring of the Easytravel Weblauncher process.
(This is specific to Easytravel)
 - Technologies > Select “weblauncher.jar” process group > Process group details > Edit > Disable for the host.



Launching the Stack

Upload Template

Filling in the Details

Create and Confirm

Follow Events

View Outputs

Validate Environment

Cleanup ◀

Cleanup

- Enable Java Hotsensor Placement
 - We will be creating Custom Services in the next section so we will need this in order to avoid a required restart of the Java Processes
 - Settings > Server-side > service monitoring > Custom service detection > Enable real-time updates to Java and PHP services

Custom service detection

Dynatrace automatically detects and monitors most server-side services in your environment with no configuration required. If your application doesn't rely on standard frameworks, you can set up custom services.

With a custom service you can instruct Dynatrace which method, class, or interface it should use to gain access to each of your application's custom server-side services.

Enable this setting to have changes to your custom services applied to Java and PHP processes in real-time with no required restart. [More...](#)

Enable real-time updates to Java and PHP services

Set to Enabled

Java services

.NET services

PHP services

Define Java services

Configuration API

Dynatrace Configuration API

- The Dynatrace Configuration API allows API consumers to globally read and write the configuration settings of any Dynatrace environment
- The primary use cases for the Configuration API are:
 - Read and copy an existing environment configuration over to a new environment.
 - Create or change individual configuration settings via the API when setting up a new environment.
 - Read and store configurations, along with their histories, in a version management system (for example, Git).
 - Provide a mechanism to automate Monitoring/Performance-as-a-Service
 - Provide easy access to meaningful data to Application Owners with creation of Management Zones
 - Simplify complex environment with auto tagging rules
 - Extract specific business context information with request attributes

View the Configuration API Explorer

- In your Dynatrace tenant navigate to:
 - Settings > Integration > Dynatrace API
 - Click "Dynatrace API Explorer"

Dynatrace API

You can use our API to export Dynatrace monitoring data into your 3rd party reporting and analysis tools. Multiple API tokens can be created for different purposes.
Use the [Dynatrace API Explorer](#) or [read the API documentation](#) for use-cases and examples.

Create API Token to use the Config API

- In the Dynatrace API screen, click “Generate token”
- Enter a name for your token (ex. MyToken)
- Make sure to enable the following switches
- Click the “Generate” button

My Dynatrace API tokens

Generate a secure access API token that enables access to your Dynatrace monitoring data via our REST-based API.

MyToken

Use the switches below to define the access scope of your Dynatrace API token.

Access problem and event feed, metrics, topology and RUM JavaScript tag management

Access logs

Configure maintenance windows

Create and configure synthetic monitors

Read configuration

Write configuration

Change data privacy settings

Capture request data

User session query language

Anonymize user session data for data privacy reasons

Log import

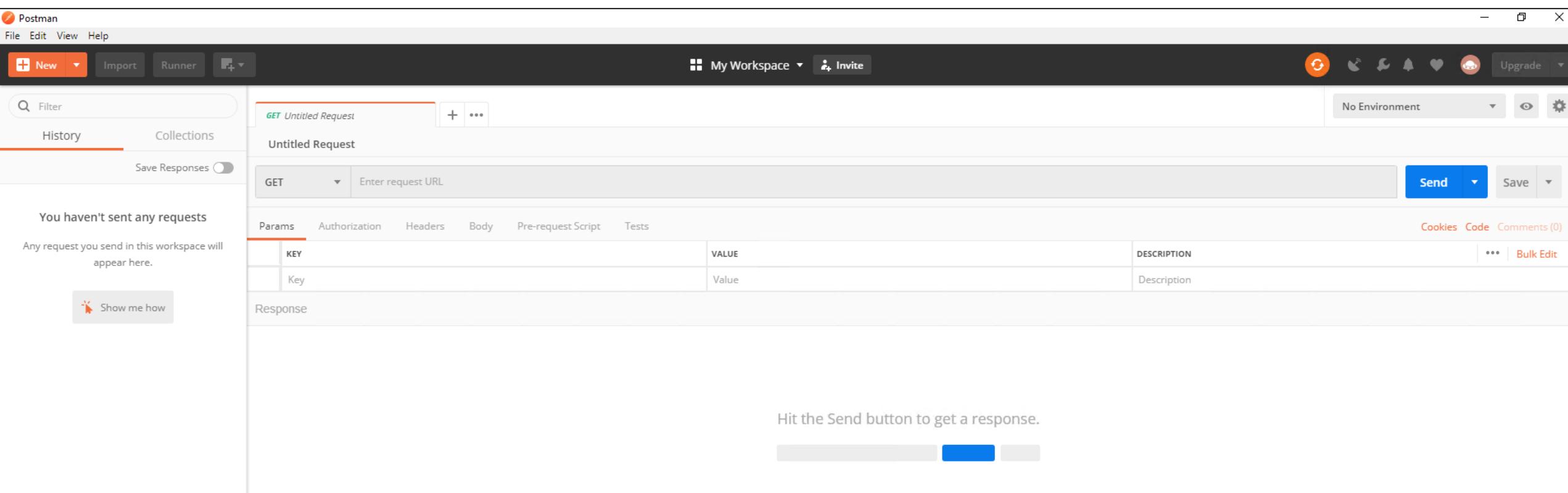
Generate **Cancel**

Using the ConfigAPI via Postman

- We will be using Postman to run a series of API requests to the ConfigAPI in order to do the following:
 - Create Auto Tagging rules
 - Create Management Zones
 - Application setup (URL Grouping and Framework support)
 - Create Custom Services
 - Create Request Attributes
 - Create Synthetics Monitors
 - Create a Dashboard
- You can use any API client to interact with any of the Dynatrace APIs, however, for this course we are using Postman because it is simple to share a collection of API requests.

Setup Postman

- If this is the first time you are using Postman, after you download and install, you will need to create a free account.
 - You can press "setup later"/"skip" on the Preferences and Team screens



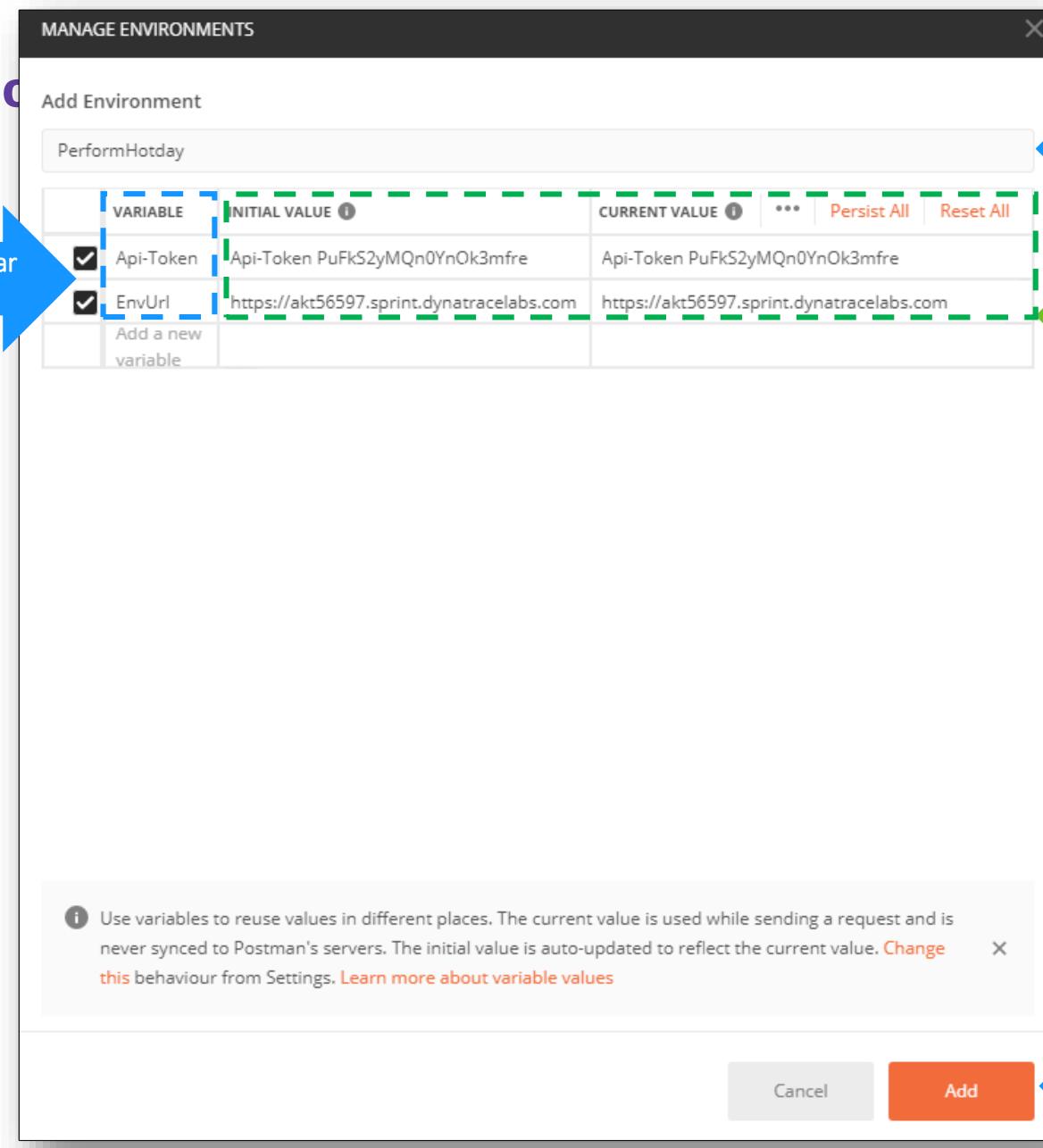
The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New' (highlighted in orange), 'Import', 'Runner', and workspace-related buttons ('My Workspace', 'Invite'). The main workspace is titled 'Untitled Request' and shows a 'GET Untitled Request' entry. The 'Params' tab is selected, displaying a table with one row:

KEY	VALUE	DESCRIPTION
Key	Value	Description

Below the table, there's a 'Response' section with the placeholder text 'Hit the Send button to get a response.' On the left sidebar, there are sections for 'History' (which is empty), 'Collections', 'Save Responses' (with a toggle switch), and a note 'You haven't sent any requests'. A 'Show me how' button is also present.

Setup Postman (cont.)

- Create new Environment
- Create 2 Variables EXACTLY as they appear here
 - CLICK the  icon



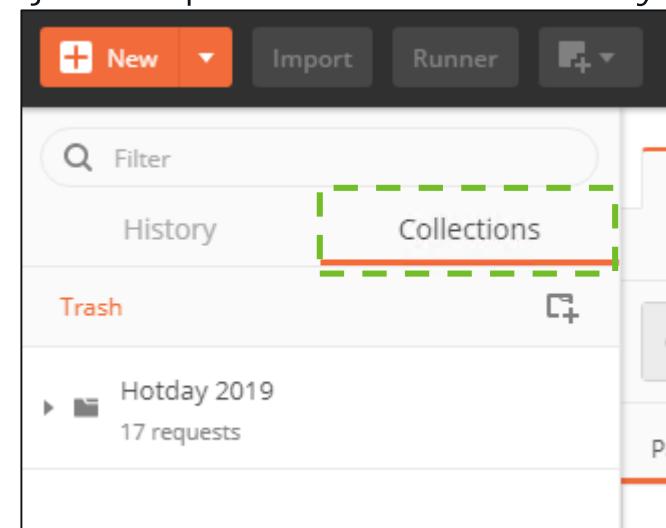
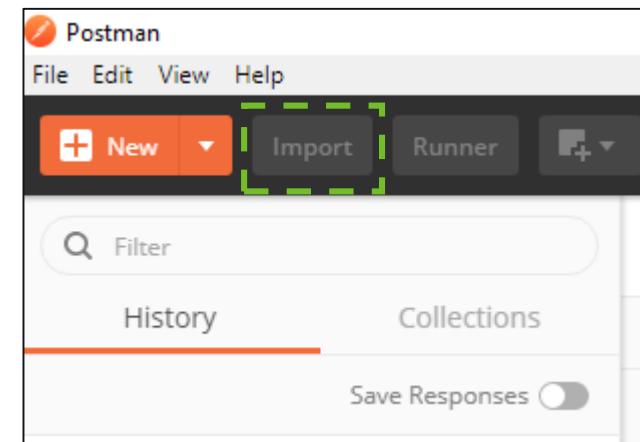
Enter a name for your Environment

Populate the values with your Full Tenant URL and API-Token

Click "Add" to create the Variables

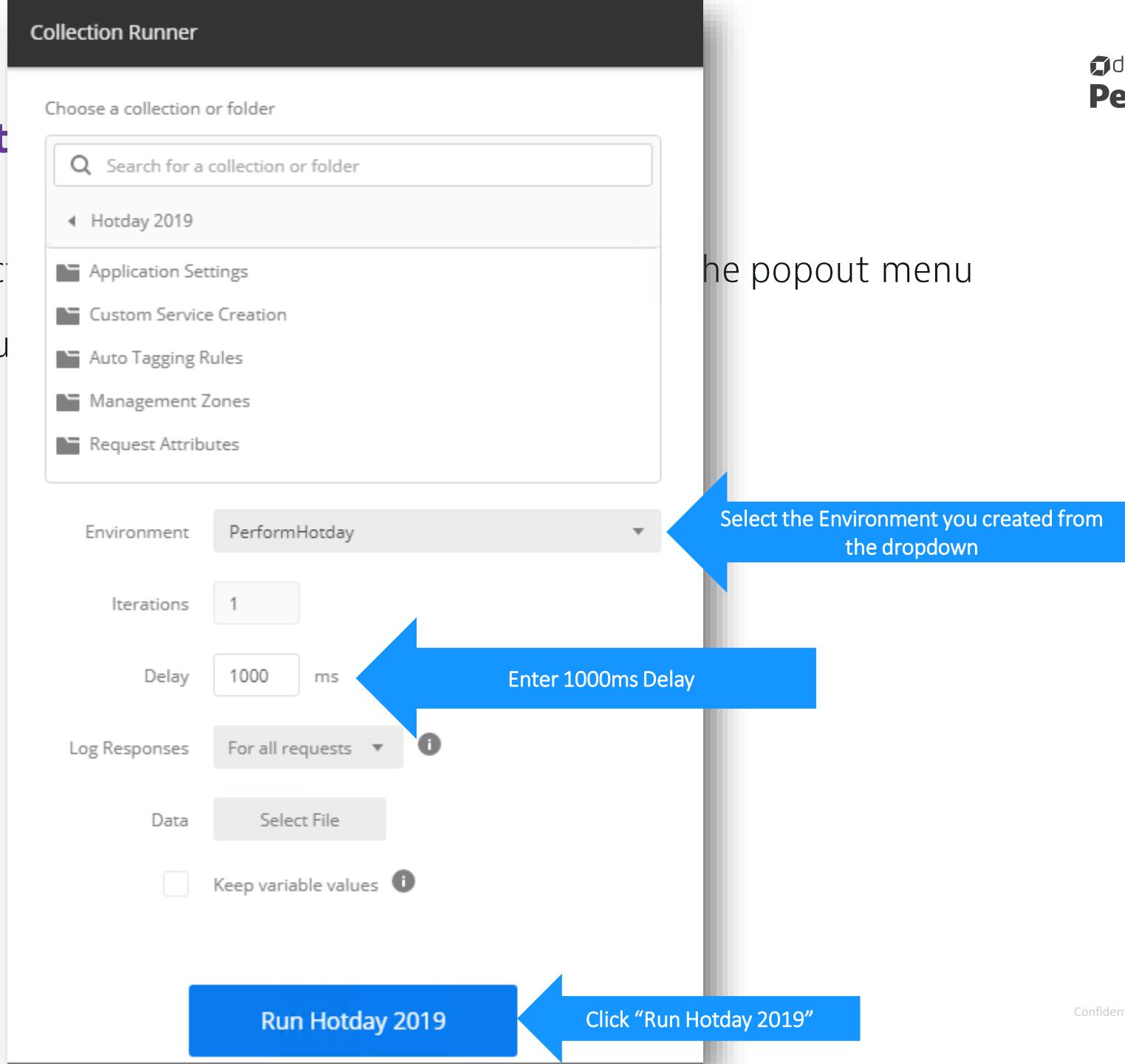
Importing the Config API Collection

- Click the “Import” button in the upper left corner
- In the Import dialog box, click “Choose Files”
- Browse to the location where you downloaded the Github repository, and select the file: “Hotday 2019.postman_collection.json”
- After the file is uploaded, select “Collections” in the upper left corner and you should now see the collect that was just imported titled “Hotday 2019”



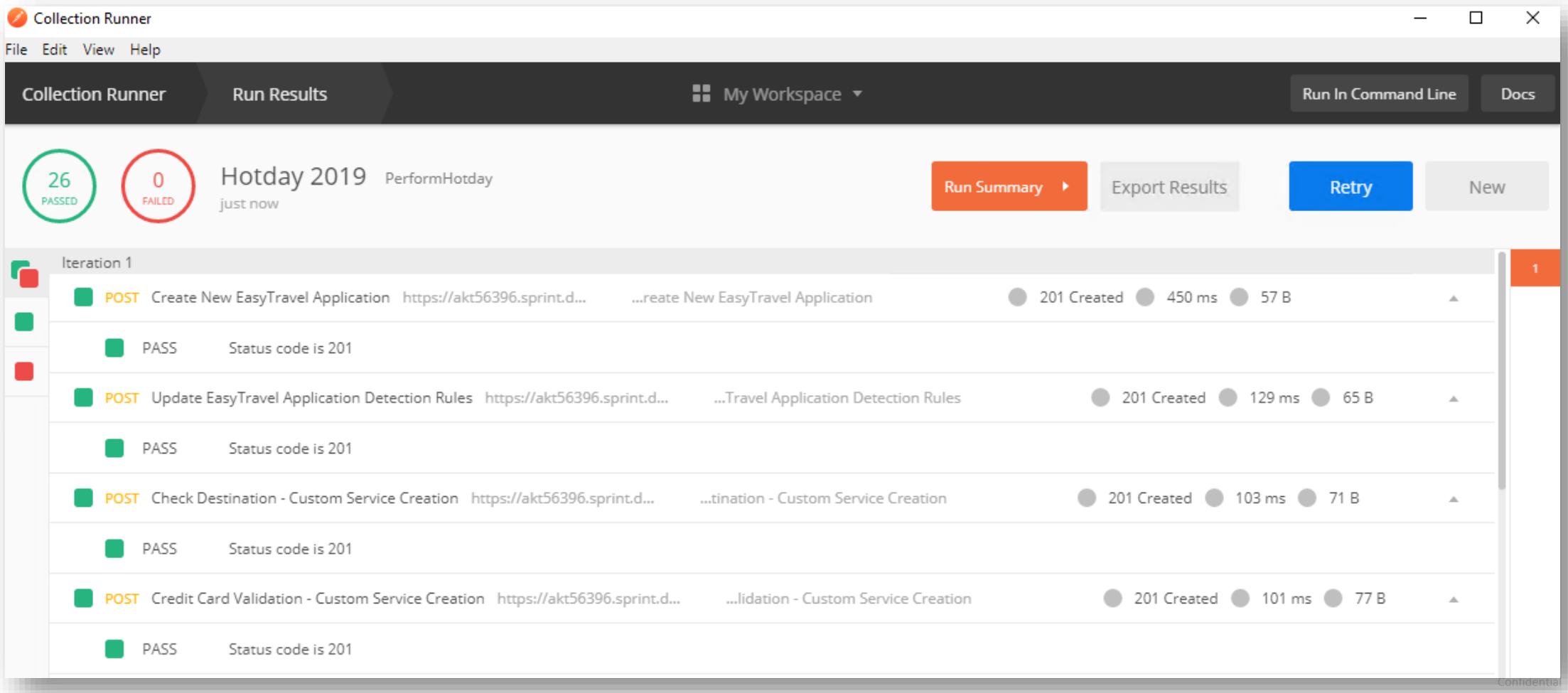
Running the API Collector

- In the “Hotday 2019” collection
- The Collection Runner should



Run Validation

- After running the collection, a Run Results window should popup with all requests successful



The screenshot shows the dynatrace Perform Collection Runner application window. The title bar reads "Collection Runner". The main menu includes "File", "Edit", "View", and "Help". The top navigation bar has tabs for "Collection Runner" (selected), "Run Results" (active), "My Workspace", "Run In Command Line", and "Docs". A status summary on the left shows "26 PASSED" in a green circle and "0 FAILED" in a red circle. The central area displays a "Hotday 2019" run from "PerformHotday" just now. It lists five successful API requests under "Iteration 1":

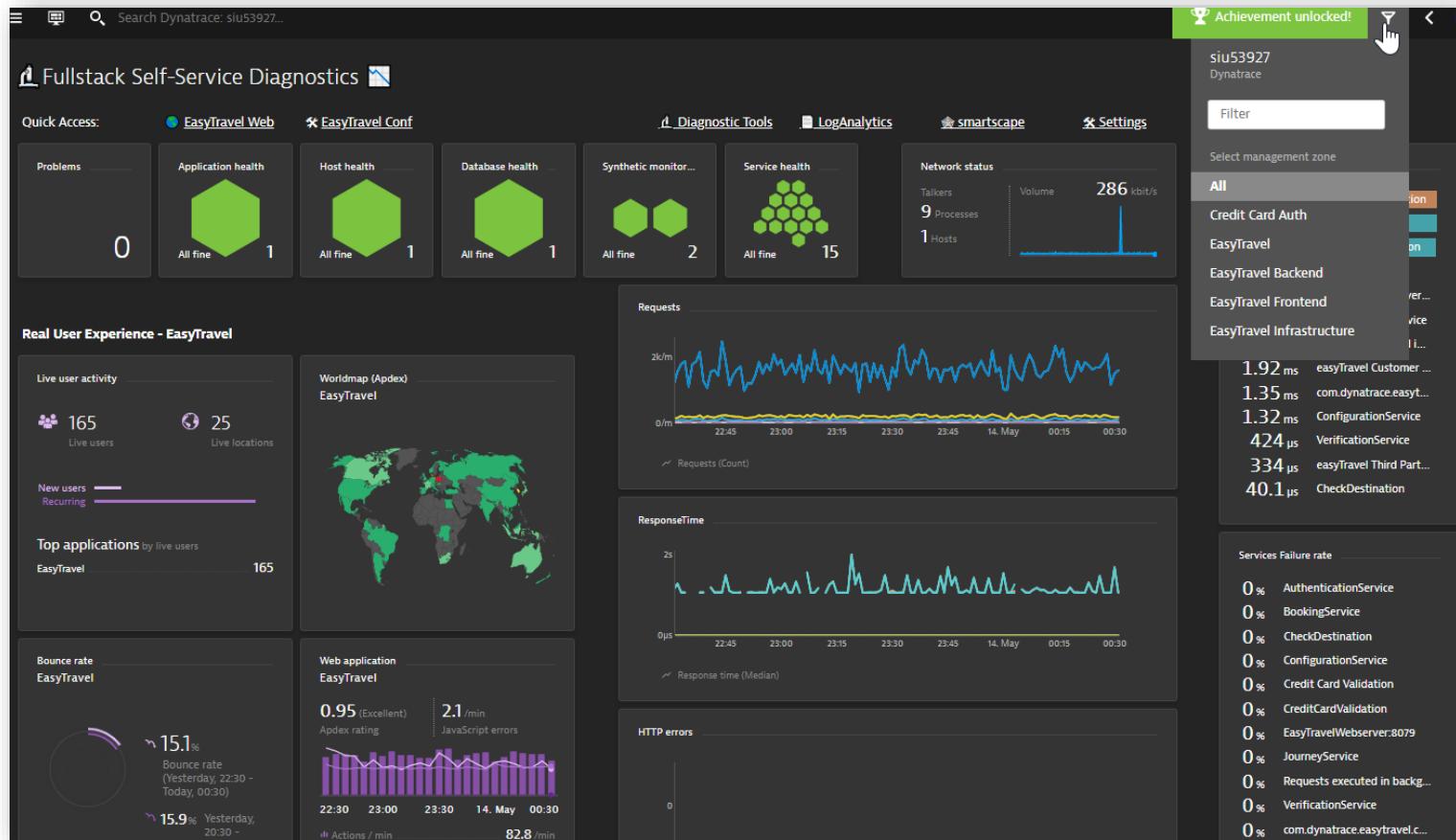
- POST Create New EasyTravel Application https://akt56396.sprint.d... ...reate New EasyTravel Application 201 Created 450 ms 57 B
- PASS Status code is 201
- POST Update EasyTravel Application Detection Rules https://akt56396.sprint.d... ...Travel Application Detection Rules 201 Created 129 ms 65 B
- PASS Status code is 201
- POST Check Destination - Custom Service Creation https://akt56396.sprint.d... ...tination - Custom Service Creation 201 Created 103 ms 71 B
- PASS Status code is 201
- POST Credit Card Validation - Custom Service Creation https://akt56396.sprint.d... ...lidation - Custom Service Creation 201 Created 101 ms 77 B
- PASS Status code is 201

Buttons for "Run Summary", "Export Results", "Retry", and "New" are visible on the right.

Request Type	Request URL	Description	Status	Time	Size
POST	Create New EasyTravel Application https://akt56396.sprint.d...	...reate New EasyTravel Application	201 Created	450 ms	57 B
PASS	Status code is 201				
POST	Update EasyTravel Application Detection Rules https://akt56396.sprint.d...	...Travel Application Detection Rules	201 Created	129 ms	65 B
PASS	Status code is 201				
POST	Check Destination - Custom Service Creation https://akt56396.sprint.d...	...tination - Custom Service Creation	201 Created	103 ms	71 B
PASS	Status code is 201				
POST	Credit Card Validation - Custom Service Creation https://akt56396.sprint.d...	...lidation - Custom Service Creation	201 Created	101 ms	77 B
PASS	Status code is 201				

Verify the Results

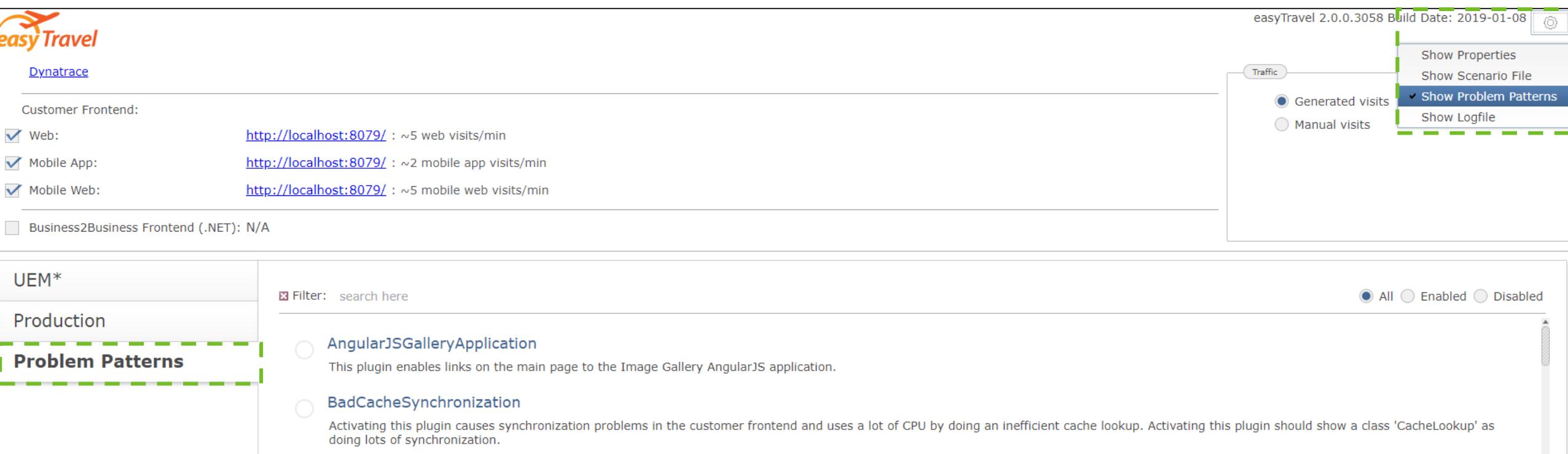
- Back in your Dynatrace tenant, you should now see the results of the Config API (ex. Management Zones, Tags, etc...)



Use Case #1 – CPU Increase

Problem Pattern #1 – CPU Increase

- How to trigger Problem:
 1. Navigate to the EasyTravelConfigUI – This URL was Output as part of the CloudFormation script
 2. Once in the EasyTravelConfigUI, enable “Show Problem Patterns”



The screenshot shows the Dynatrace interface for the 'easyTravel' application. At the top right, it displays the version 'easyTravel 2.0.0.3058 Build Date: 2019-01-08'. Below this is a context menu with options: 'Show Properties', 'Show Scenario File', 'Show Problem Patterns' (which is checked), and 'Show Logfile'. The main left panel shows 'Customer Frontend' traffic with three selected items: 'Web' (localhost:8079), 'Mobile App' (localhost:8079), and 'Mobile Web' (localhost:8079). The 'Business2Business Frontend (.NET)' is listed as 'N/A'. On the right, the 'Problem Patterns' section is visible, containing two entries: 'AngularJSGalleryApplication' and 'BadCacheSynchronization'. A sidebar on the left includes tabs for 'UEM*', 'Production', and 'Problem Patterns'.

easyTravel 2.0.0.3058 Build Date: 2019-01-08

Show Properties
Show Scenario File
Show Problem Patterns
Show Logfile

Customer Frontend:

- Web: <http://localhost:8079/> : ~5 web visits/min
- Mobile App: <http://localhost:8079/> : ~2 mobile app visits/min
- Mobile Web: <http://localhost:8079/> : ~5 mobile web visits/min

Business2Business Frontend (.NET): N/A

UEM*

Production

Problem Patterns

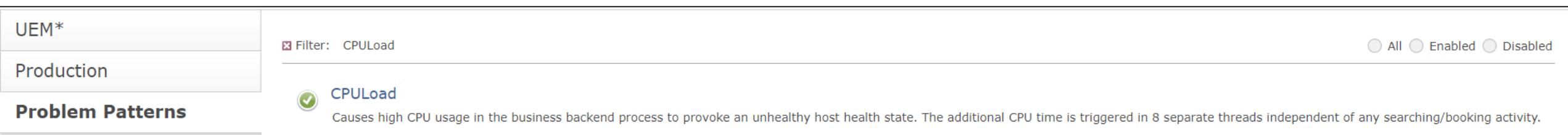
Filter: search here

All Enabled Disabled

- AngularJSGalleryApplication**
This plugin enables links on the main page to the Image Gallery AngularJS application.
- BadCacheSynchronization**
Activating this plugin causes synchronization problems in the customer frontend and uses a lot of CPU by doing an inefficient cache lookup. Activating this plugin should show a class 'CacheLookup' as doing lots of synchronization.

Problem Pattern #1 – CPU Increase

- How to trigger Problem:
 1. In the Filter bar search for the Problem Pattern "CPULoad"
 2. Click the "CPULoad" Problem Pattern to trigger the problem



The screenshot shows the Dynatrace interface with the following details:

- UEM*** is selected in the navigation bar.
- Production** is selected in the environment dropdown.
- Problem Patterns** is selected in the main menu.
- Filter: CPULoad** is applied, indicated by a checked checkbox icon.
- All**, **Enabled**, and **Disabled** buttons are available for filtering.
- CPULoad** is listed as a problem pattern, marked with a green checkmark.
- A tooltip for CPULoad states: "Causes high CPU usage in the business backend process to provoke an unhealthy host health state. The additional CPU time is triggered in 8 separate threads independent of any searching/booking activity."

Use Case #1 – CPU Increase

Background

EasyTravel is the main revenue generating application for booking travel. After a recent deployment to one of the backend components, your team notices that CPU usage is much higher than usual. This of course could impact users attempting to book travel on our site.

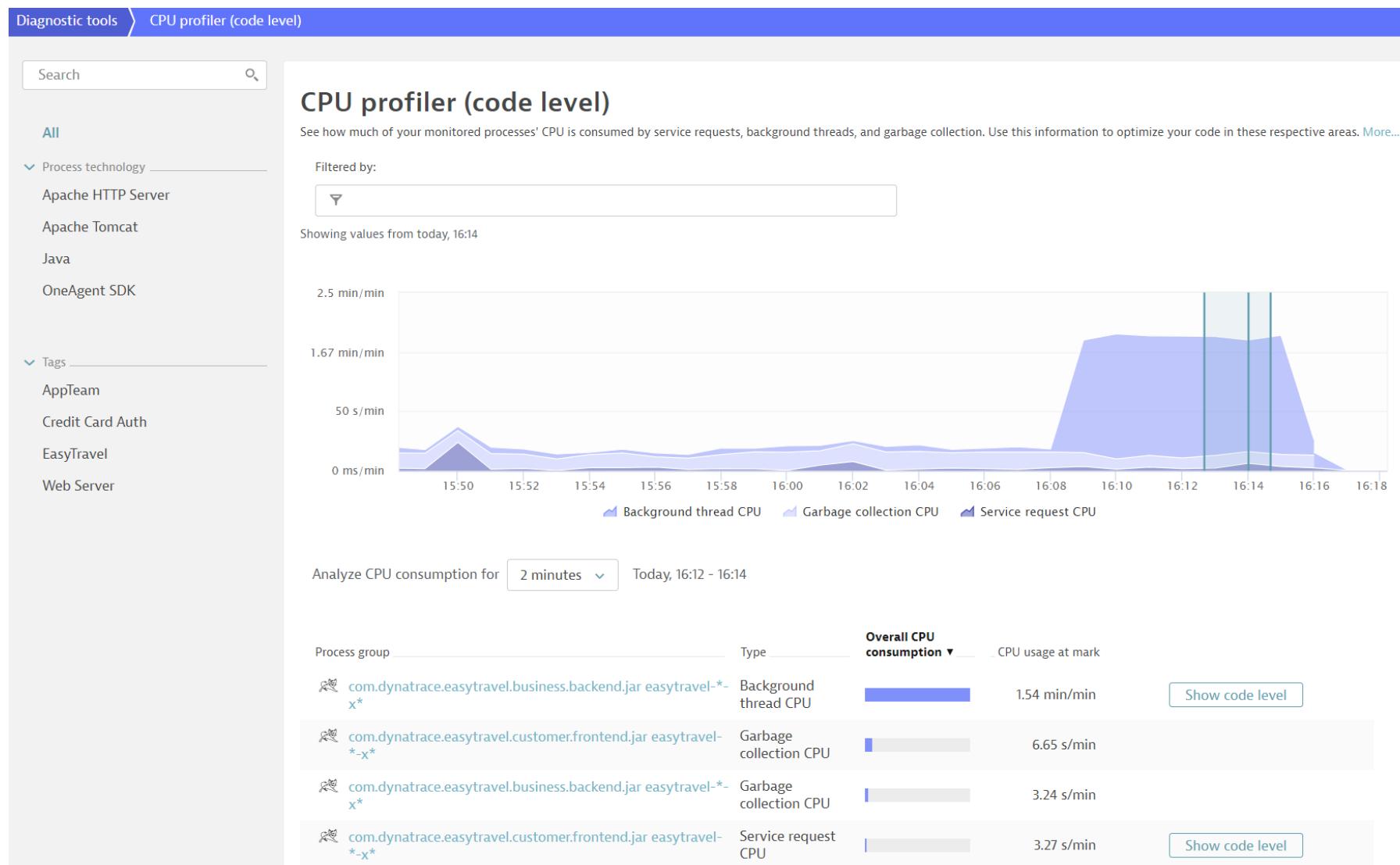
1. Investigate and figure out why this is happening
2. How would you instruct the Dev team to go about finding the problem and triaging it?

Use Case #1 – CPU Increase (cont.)

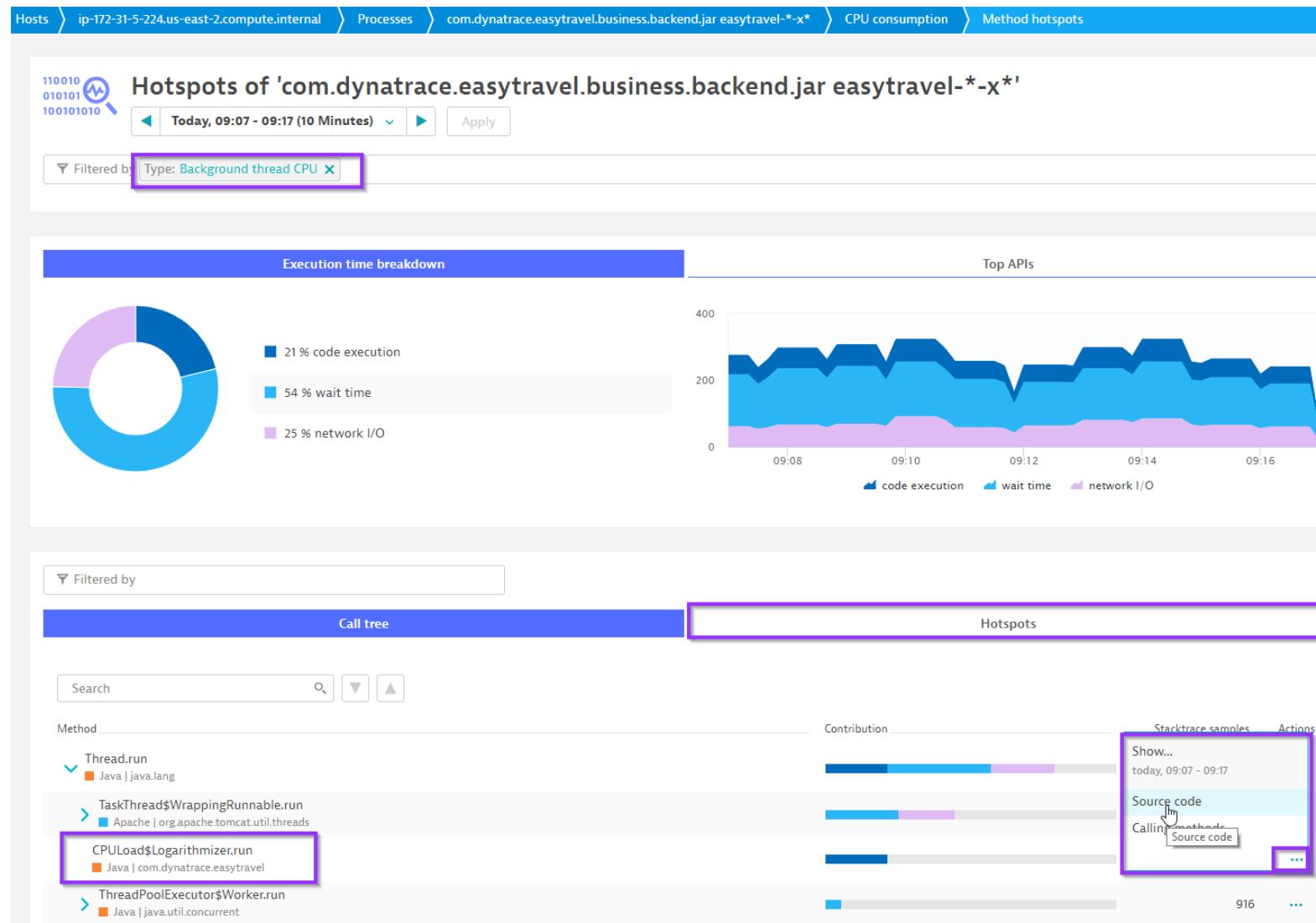
How to investigate the problem:

- Navigate to Diagnostic Tools and select the CPU Profiler (filter by Management Zone or Tag)
- View the Service with the Highest CPU consumption
- Navigate to the code level breakdown to determine what piece of code is causing increase in CPU consumption

Use Case #1 – CPU Increase (cont.)



Use Case #1 – CPU Increase (cont.)



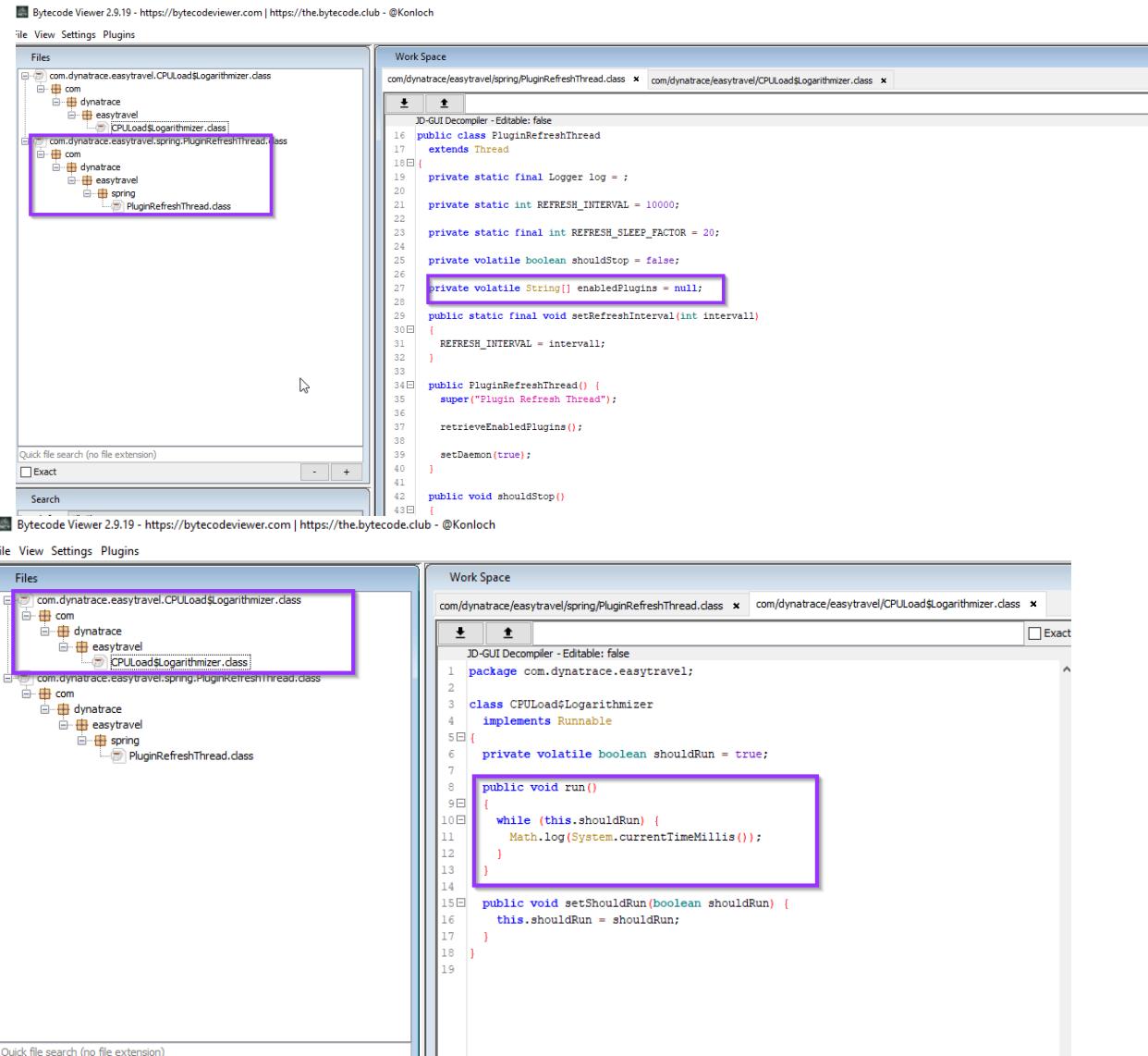
Use Case #1 – CPU Increase (ext.)

This is an excuse. Continue if you were super fast and we have not started UseCase 2 continue.

This is a way to show that you can analyze even deeper to the bits und bytes as a developer.

If you decompile the CPULoadLogarithmizer you'll see that the CPU increases due calculating the Logarithm of the current time in Miliseconds.

The Plugin is loaded and started as a Thread in the class PluginRefreshThread (also found in the Method Hotspots)



```

Bytecode Viewer 2.9.19 - https://bytecodeviewer.com | https://the.bytecode.club - @Konloch
File View Settings Plugins
Files
com.dynatrace.easytravel.CPULoad$Logarithmizer.class
  + com
    + dynatrace
      + easytravel
        + CPULoad$Logarithmizer.class
com.dynatrace.easytravel.spring.PluginRefreshThread.class
  + com
    + dynatrace
      + easytravel
        + spring
          + PluginRefreshThread.class
Work Space
JD-GUI Decompiler - Editable: false
public class PluginRefreshThread
extends Thread
{
    private static final Logger log = ;
    private static int REFRESH_INTERVAL = 10000;
    private static final int REFRESH_SLEEP_FACTOR = 20;
    private volatile boolean shouldStop = false;
    private volatile String[] enabledPlugins = null;
    public static final void setRefreshInterval(int interval)
    {
        REFRESH_INTERVAL = interval;
    }
    public PluginRefreshThread()
    {
        super("Plugin Refresh Thread");
        retrieveEnabledPlugins();
        setDaemon(true);
    }
    public void shouldStop()
    {
    }
}

Bytecode Viewer 2.9.19 - https://bytecodeviewer.com | https://the.bytecode.club - @Konloch
File View Settings Plugins
Files
com.dynatrace.easytravel.CPULoad$Logarithmizer.class
  + com
    + dynatrace
      + easytravel
        + CPULoad$Logarithmizer.class
com.dynatrace.easytravel.spring.PluginRefreshThread.class
  + com
    + dynatrace
      + easytravel
        + spring
          + PluginRefreshThread.class
Work Space
JD-GUI Decompiler - Editable: false
package com.dynatrace.easytravel;
class CPULoad$Logarithmizer
implements Runnable
{
    private volatile boolean shouldRun = true;
    public void run()
    {
        while (this.shouldRun)
        {
            Math.log(System.currentTimeMillis());
        }
    }
    public void setShouldRun(boolean shouldRun)
    {
        this.shouldRun = shouldRun;
    }
}

```

Use Case #1 – CPU Increase (ext.)

Alternatively you can see when the CPU Plugin was enabled in the Log files. By searching on the Logs of the processes and the Host with the Query *CPU* OR Plugin*.

Search

Use our [query language](#) to search for text patterns inside the selected log files.

★

Selected log files:

6 logs on 5 processes running on ip-172-31-5-224.us-east-2.compute.internal selected (15.6 MB) Change selection

Display results Download log files Define event

Results

View chart by source Export results

View chart by source

2019-05-17 08:59

ip-172-31-5-224.us...2.compute.internal : /root/.dynaTrace/easyTravel ...avel/log/BusinessBackend.log 2
ip-172-31-5-224.us...2.compute.internal : /var/log/syslog 4

08:10 08:15 08:20 08:25 08:30 08:35 08:40 08:45 08:50 08:55 09:00 09:05 09:10 09:15 09:20 09:25 09:30 09:35 09:40 09:45

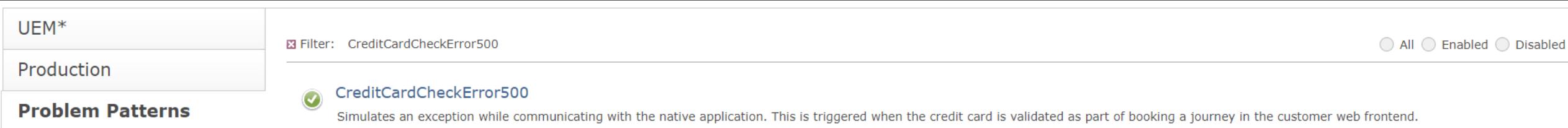
ip-172-31-5-224.us...2.compute.internal : /root/.dynaTrace/easyTravel ...avel/log/BusinessBackend.log ip-172-31-5-224.us...2.compute.internal : /var/log/syslog

#	Level	Count	Source	Signature
1	WARN	4		*2019-05-17 06:59:46 BusinessBa WARN [PluginEventNotifier] Multiple templates for the same plugin name: REPLACEME
2	INFO	5		*2019-05-17 07:00:* BusinessBa INFO [PluginStateManager] *abled plugin: CPULoad
				May 17 07:00:17 ip-172-31-5-224 cloud-init[1377]: 2019-05-17 07:00:17 BusinessBa INFO [PluginStateManager] enabled plugin: CPULoad
				2019-05-17 07:00:17 BusinessBa INFO [PluginStateManager] enabled plugin: CPULoad
				May 17 07:00:19 ip-172-31-5-224 cloud-init[1377]: 2019-05-17 07:00:19 BusinessBa INFO [PluginStateManager] disabled plugin: CPULoad
				May 17 07:00:23 ip-172-31-5-224 cloud-init[1377]: 2019-05-17 07:00:23 BusinessBa INFO [PluginStateManager] enabled plugin: CPULoad

Use Case #2 – Booking Failure

Problem Pattern #2 – Booking Failure

- How to trigger Problem:
 1. In the Filter bar search for the Problem Pattern "CreditCardCheckError500"
 2. Click the "CreditCardCheckError500" Problem Pattern to trigger the problem



The screenshot shows the Dynatrace User Experience Management (UEM) interface. On the left, there's a sidebar with tabs for 'UEM*' (selected), 'Production' (disabled), and 'Problem Patterns'. The 'Problem Patterns' tab is active, showing a list of patterns. One pattern is selected: 'CreditCardCheckError500'. A tooltip for this pattern states: 'Simulates an exception while communicating with the native application. This is triggered when the credit card is validated as part of booking a journey in the customer web frontend.' At the top of the main content area, there's a filter bar with the text 'Filter: CreditCardCheckError500' and three status buttons: 'All' (selected), 'Enabled', and 'Disabled'.

Use Case #2 – Booking Failure

Background

The Marketing Department just called in a panic that bookings dropped to near zero over the past few minutes! Obviously, this is a pretty big issue because the core business is being impacted.

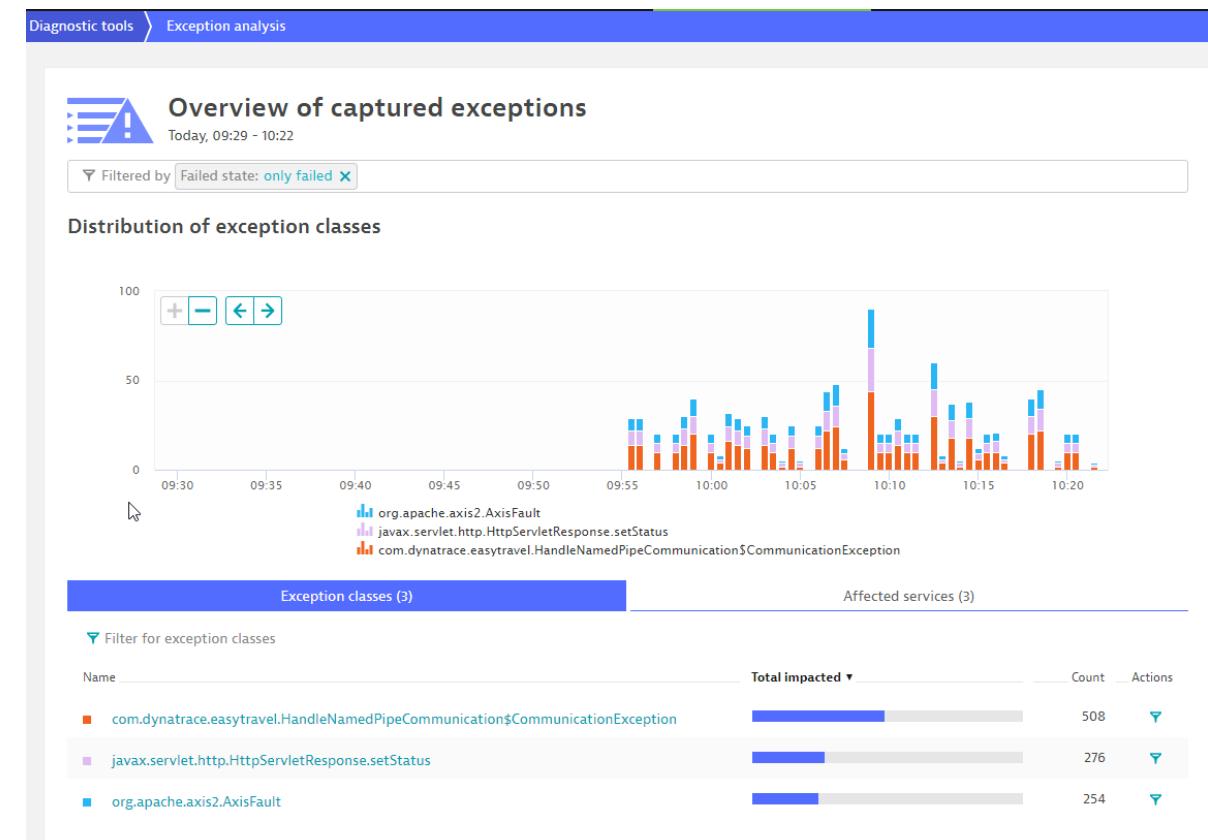
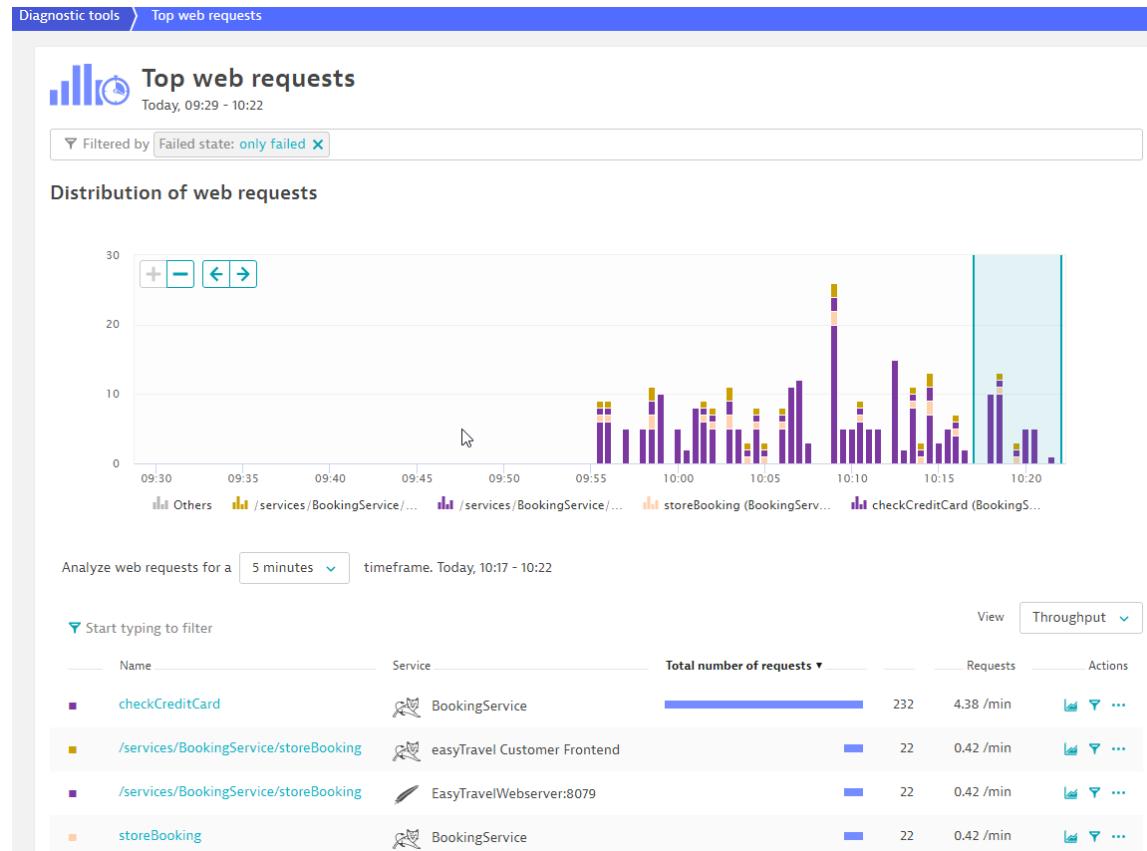
1. Investigate and figure out why this is happening
2. How would you instruct the Dev team to go about finding the problem and triaging it?

Use Case #2 – Booking Failure (cont.)

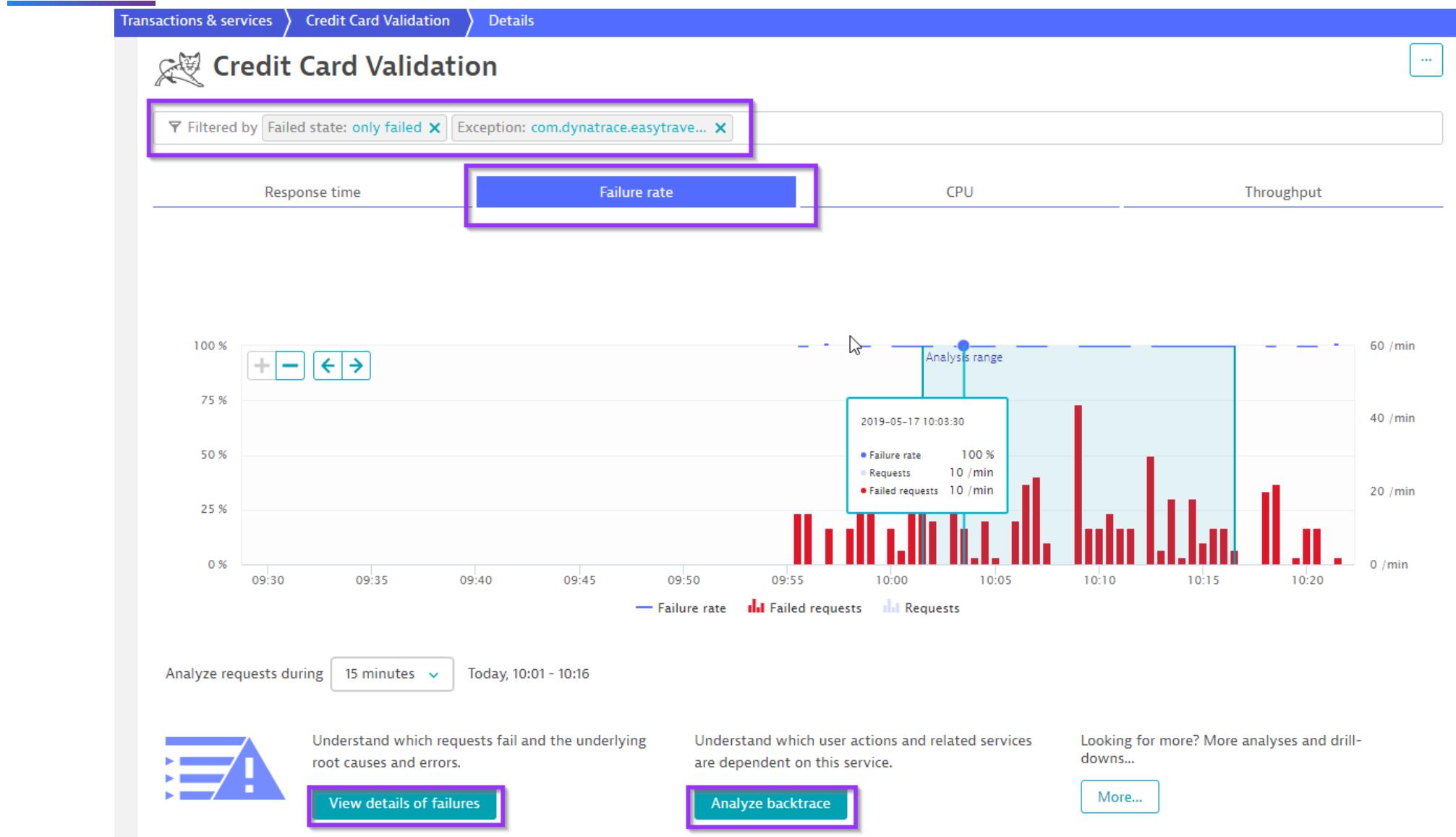
How to investigate the problem:

- Navigate to Diagnostic Tools and select either:
 - Exception Analysis (filter by failedState:onlyFailed)
 - OR Top Web Requests (filter by failedState:onlyFailed)
- Determine which Service is affected
- Investigate the details of the Exceptions/Failed Requests and what should be done
- Find out quickly what is impacted by the Exception/Requests (use "Backtrace" feature)

Use Case #2 – Booking Failure (cont.)



Use Case #2 – Booking Failure (cont.)



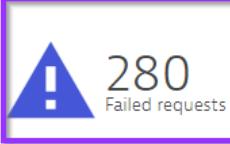
Use Case #2 – Booking Failure (cont.)

Transactions & services > Credit Card Validation > Details > Failure analysis

Failed request of 'Credit Card Validation'

Today, 10:00 - 10:30 (30 Minutes) ▶ Apply

Filtered by Request type: Default requests Failed state: only failed



280
Failed requests

Reasons for failed requests
Following are the reasons for failed requests. Click a specific reason to view the exception message and code-level details.

Reason	Failed requests
com.dynatrace.easytravel.HandleNamedPipeCommunication\$CommunicationException / Communication plugin could not contact credit card verification application via named pipe.	100 % 280

Requests that failed with com.dynatrace.easytravel.HandleNamedPipeCommunication\$CommunicationException / Communication plugin could not contact credit card verification application via named pipe.

Reason	Failed requests
SocketNativeApplication.sendAndReceive	280

Root causes for requests that failed with com.dynatrace.easytravel.HandleNamedPipeCommunication\$CommunicationException / Communication plugin could not contact credit card verification application via named pipe.

100 % were caused by com.dynatrace.easytravel.HandleNamedPipeCommunication\$CommunicationException

Show potential root causes Details

Confidential

Use Case #2 – Booking Failure (cont.)

Transactions & services > Credit Card Validation > Details > Backtrace

Service-level backtrace of requests of 'Credit Card Validation'

Today, 10:01 - 10:16 | Apply

Filtered by Exception: com.dynatrace.easytrave... | Failed state: only failed

The services and applications listed below make calls to this service. The tree view represents the sequence of services and application user actions that led to this service call, beginning with the page load or user action in the browser that triggered the sequence. Click to see which specific requests and user actions called this service

Incoming requests to this service

- Credit Card Validation
 - BookingService
 - easyTravel Customer Frontend
 - EasyTravelWebserver:8079
 - Apache Web Server easyTravel
- EasyTravel
 - Web application

Requests from EasyTravelWebserver:8079

Requests	Instances
/orange-booking-payment.jsf	147 Requests 0 Failed requests
/services/BookingService/storeBooking	16 Requests 16 Failed requests

Analysis of EasyTravelWebserver:8079 requests



- 163 analyzed EasyTravelWebserver:8079 requests trigger 163 calls to easyTravel Customer Frontend
- 16 EasyTravelWebserver:8079 requests failed
- 16 calls to easyTravel Customer Frontend failed, potentially impacting EasyTravelWebserver:8079 requests

Reasons for failed requests Stacktrace Referring pages Proxies Analyze

For root cause go to 'Analyze' tab and click 'View details of failures' of selected service.

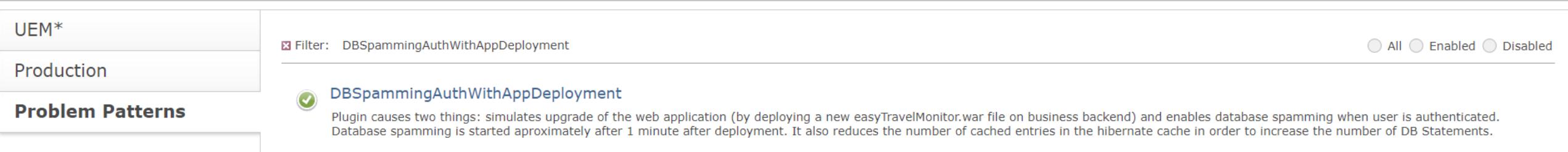
> HTTP response code 5xx

16 Failed requests

Use Case #3 – Slow User Login

Problem Pattern #3 – Slow User Login

- How to trigger Problem:
 1. In the Filter bar search for the Problem Pattern "DBSpammingAuthWithAppDeployment"
 2. Click the "DBSpammingAuthWithAppDeployment" Problem Pattern to trigger the problem



The screenshot shows the Dynatrace User Experience Management (UEM) interface. On the left, there's a sidebar with tabs for 'UEM*' (selected), 'Production', and 'Problem Patterns'. The 'Problem Patterns' tab is currently active. In the main content area, there's a filter bar at the top with a checked checkbox labeled 'Filter: DBSpammingAuthWithAppDeployment'. To the right of the filter are three radio buttons labeled 'All', 'Enabled', and 'Disabled'. Below the filter, a list item is shown with a green checkmark icon and the text 'DBSpammingAuthWithAppDeployment'. A detailed description follows: 'Plugin causes two things: simulates upgrade of the web application (by deploying a new easyTravelMonitor.war file on business backend) and enables database spamming when user is authenticated. Database spamming is started approximately after 1 minute after deployment. It also reduces the number of cached entries in the hibernate cache in order to increase the number of DB Statements.'

Use Case #3 – Slow User Login

Background

After a recent deployment to a component of EasyTravel customers are beginning to call the helpdesk and complain that logging in to view/book travel is “slow”.

1. Investigate and figure out why this is happening
2. How would you instruct the Dev team to go about finding the problem and triaging it?

Use Case #3 – Slow User Login (cont.)

How to investigate the problem:

- Analyze the User Action "click on "Login" on page /orange.jsf"
- View the response time breakdown and it's service flow
- Investigate the "AuthenticationService" and determine where the response time originates from
- Compare the current response time with the previously seen response times (using comparison feature)
- Is this a Database Slowdown issue?

Use Case #3 – Slow User Login (cont.)

Applications > EasyTravel > User actions > click on "Login" on page /orange.jsf

Data for non-key user actions is available back to 03 April 2019 10:10 AM. For longer historical trends, setting a user action specific Apdex threshold or adding a user action tile to your dashboard, mark this user action as **key user action** by using the context button.



1.1 s User action duration
4.6 /min User actions



0.0 /min JavaScript errors



1.00 (Excellent) Apdex rating

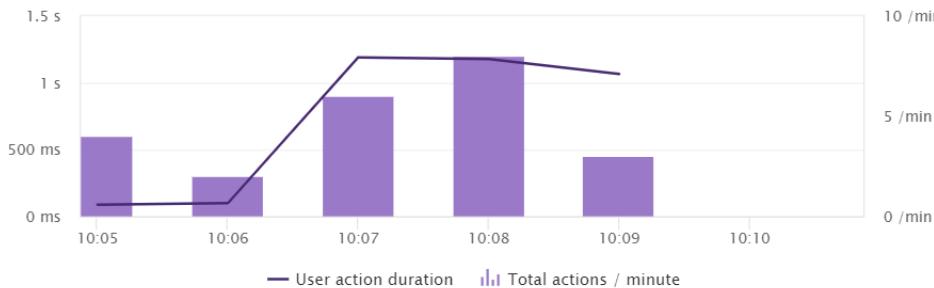


0.0 Resources /action
0.9 CDN and 3rd party resources

Performance

View the impact of user activity on performance, check out the various performance contributors and see the distribution for your selected performance metric.

Show Median Slowest 10% Fastest 10% for Key performance metric



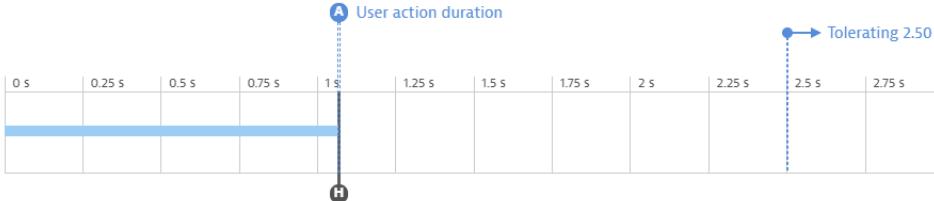
The chart displays two metrics over time from 10:05 to 10:10. The Y-axis shows User action duration in seconds (ms) and Total actions / minute. The X-axis shows the time period. The User action duration bar chart shows values around 500 ms, 800 ms, 1.2 s, and 400 ms. The Total actions / minute line graph shows a sharp increase from 0 to approximately 10/min between 10:07 and 10:08.

[Show contributors and distribution](#)

No problems Today, 10:05 - 10:10

Contributors breakdown

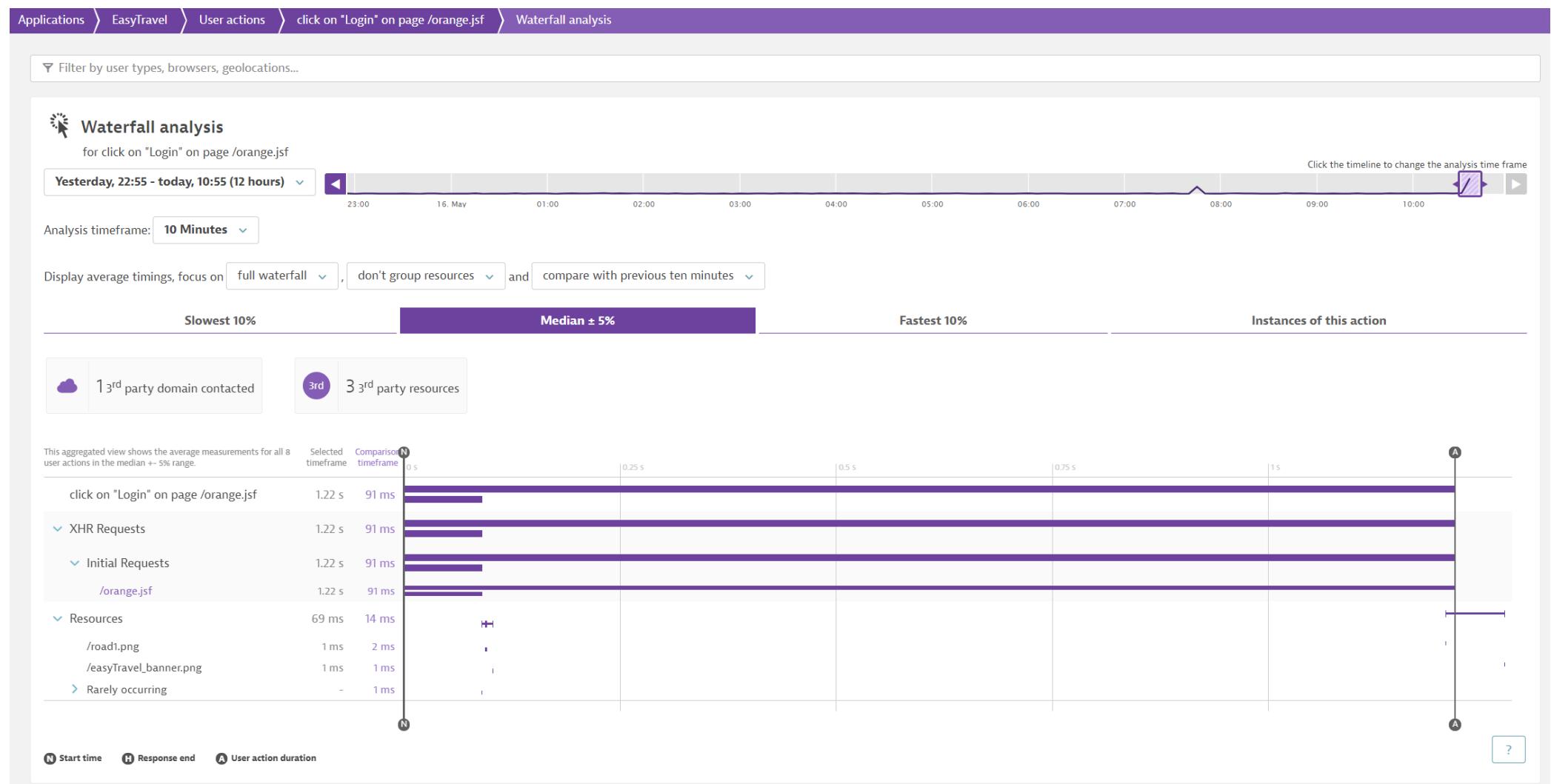
Network/Server/Frontend breakdown based on W3C navigation timings. Open the waterfall view for full details.



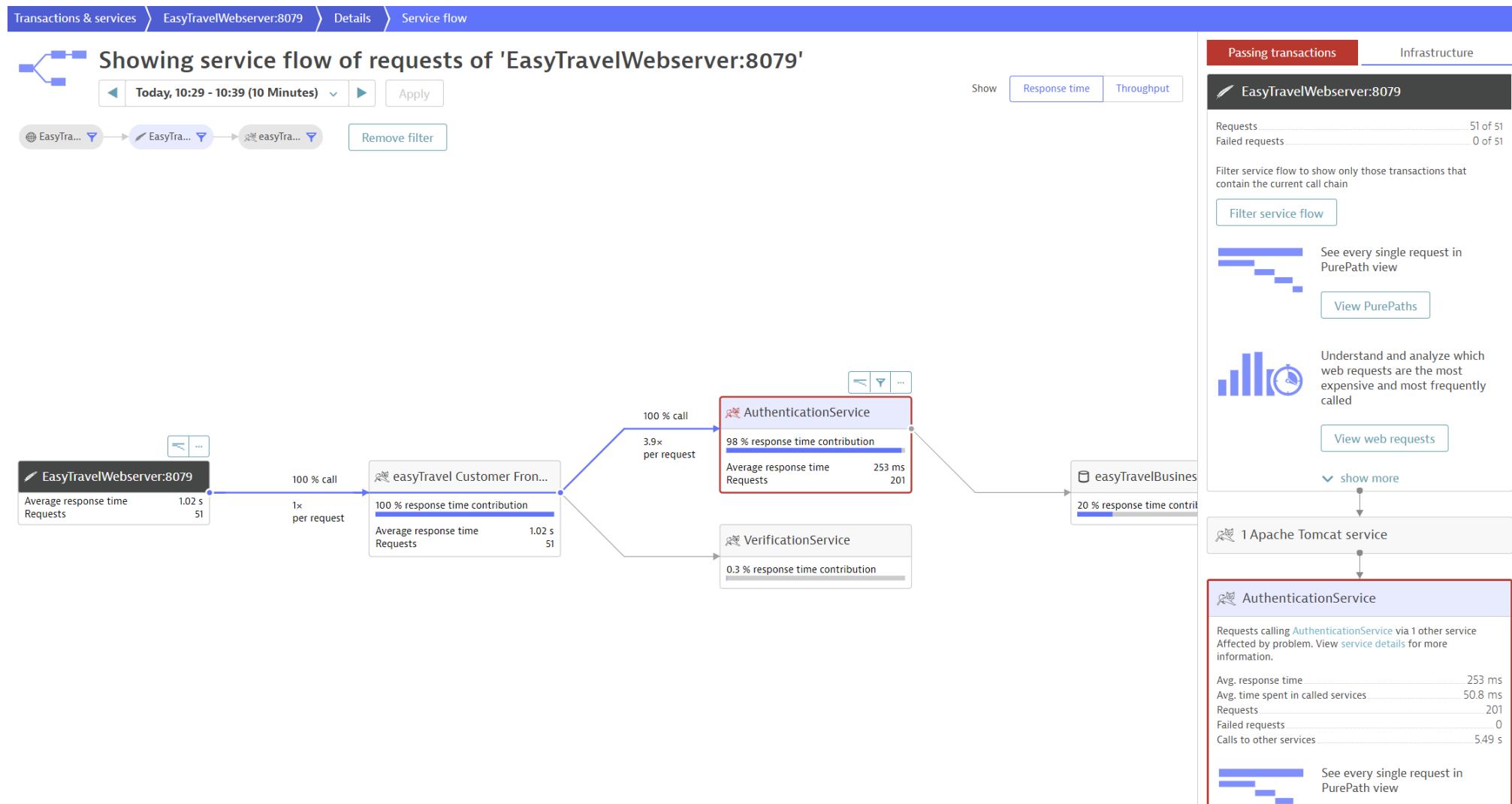
The waterfall chart shows the breakdown of navigation times. It includes Frontend time, Network time, and Server time. A vertical dashed line at 1.1s is labeled "User action duration". Another vertical dashed line at 2.50s is labeled "Tolerating 2.50 s". A horizontal bar spans from 0.25s to 1.07s, ending at a marker labeled "Response end 1.07 s".

[View analysis in waterfall chart](#)

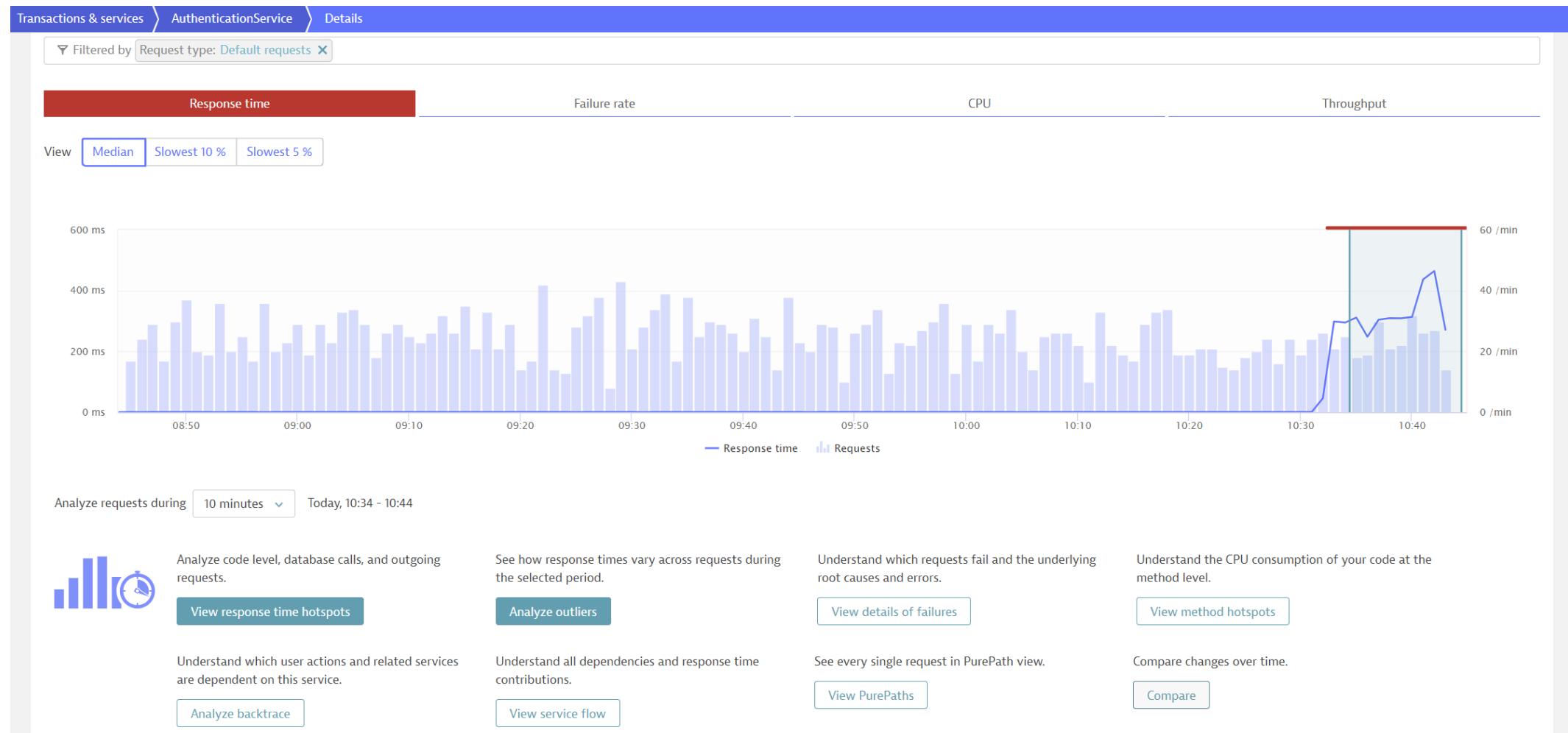
Use Case #3 – Slow User Login (cont.)



Use Case #3 – Slow User Login (cont.)



Use Case #3 – Slow User Login (cont.)



Use Case #3 – Slow User Login (cont.)

Transactions & services > AuthenticationService > Details > Response time analysis

Distribution

- No interaction with services or queues
- Database usage: 445 ms
- Service execution: 445 ms

Top findings

Event Type	Description	Time Delta
Code execution	+216 ms	
Suspension	+104 ms	
SQL Query	select ... from LoginUser user0_order by user0_n ... in SQL Queries or Procedures +75.5 ms	

select ... from LoginUser user0_order by user0_n ... in SQL Queries or Procedures

AuthenticationService ... → 100% → 1.06x

Metric	Value	Delta
Response time	70.9 ms	+70.9 ms
Percentage of calls	100 %	+100 %
Invocations per call	1.06 x	+1.06 x
= Response time contribution	75.5 ms	+75.5 ms

100% of the requests call `select ... from LoginUser user0_order by user0_n ...` (averaging 1.06 calls per request).

Database statements of select ... from LoginUser user0_order by user0_n ...

Statement	Contribution
<pre>select user0_name as name1_5_, user0_email as email2_5_ user0_fullName as fullName3_5_ user0_lastLogin as lastLogIn4_5_ user0_loyaltyStatus as loyalty5_5_ user0_password as password6_5_ from LoginUser user0_order by user0_name</pre>	75.5 ms

Use Case #4 – Slow Loading Recommendations

Problem Pattern #4 – Slow Loading Recommendations

- How to trigger Problem:
 1. In the Filter bar search for the Problem Pattern "DatabaseAccessPoolContentionSync"
 2. Click the "DatabaseAccessPoolContentionSync" Problem Pattern to trigger the problem

Problem Patterns

'by design' only the business backend application should do. Also uses a non-connection pool connection which is displayed differently in the Database Dashlet. Look for class 'QueryLocations' in the PurePath. Note: This plugin puts a heavy load on CPU as well, not suited for live demos!

 [DatabaseAccessPoolContention](#)

On the Business Backend application, while searching Locations, executes a stored procedure that contends a db-pool for a few seconds. The effect in dynaTrace will be that you see high database calls of a stored procedure which allows to show database related use cases. The database call will be done asynchronously and will show a database time of 2 seconds.

 [DatabaseAccessPoolContentionSync](#)

On the Business Backend application, while searching Locations, executes a stored procedure that contends a db-pool for a few seconds. The effect in dynaTrace will be that you see high database calls of a stored procedure which allows to show database related use cases. This will do the call synchronously as part of the transaction and will cause a database delay of 500 milliseconds.

Use Case #4 – Slow Loading Recommendations

Background

Your shop manager is calling you up and complaining that booking recommendations on the page are taking an unusually long time to show up.

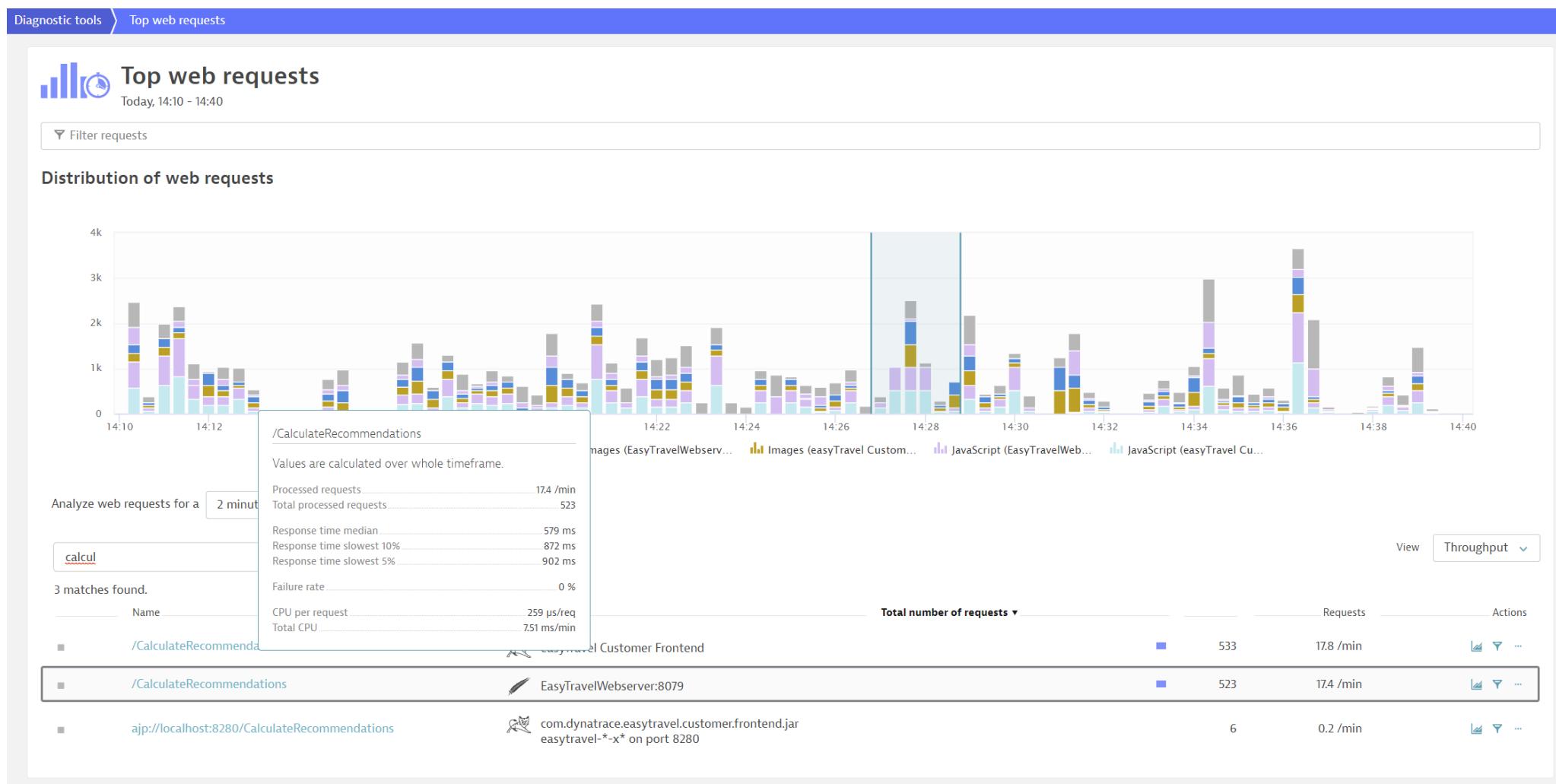
1. Investigate and figure out why this is happening
2. How would you instruct the Dev team to go about finding the problem and triaging it?

Use Case #4 – Slow Loading Recommendations (cont.)

How to investigate the problem:

- In Diagnostic Tools, view the Top Web Requests and search for “CalculateRecommendations” Web Requests
- View the Service Flow and look for the highest % of contribution
- Use the compare feature to analyze the degradation
- Use the backtrace feature to find the impact on the frontend

Use Case #4 – Slow Loading Recommendations (cont.)

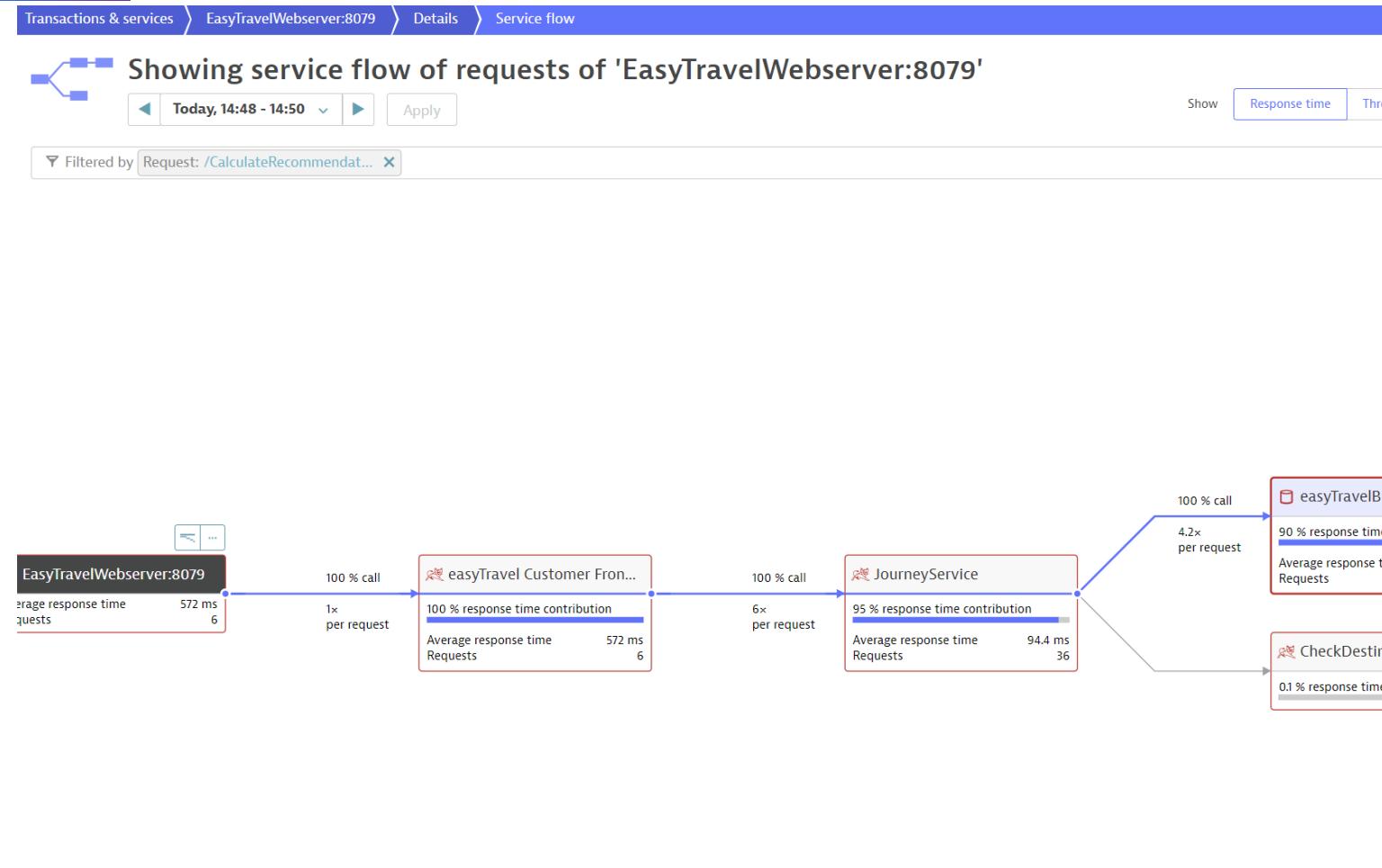


Use Case #4 – Slow Loading Recommendations (cont.)

Transactions & services > EasyTravelWebserver:8079 > Details > Service flow

Showing service flow of requests of 'EasyTravelWebserver:8079'

Today, 14:48 - 14:50 | Apply | Filtered by Request: /CalculateRecommendat... | Show Response time Throughput



Service	Response Time Contribution	Avg. Response Time (ms)	Requests
EasyTravelWebserver:8079	100 %	572 ms	6
easyTravel Customer Frontend	100 %	572 ms	6
JourneyService	95 %	94.4 ms	36
easyTravelBusiness	90 %	21.4 ms	150
CheckDestination	0.1 %	0 ms	0

Passing transactions

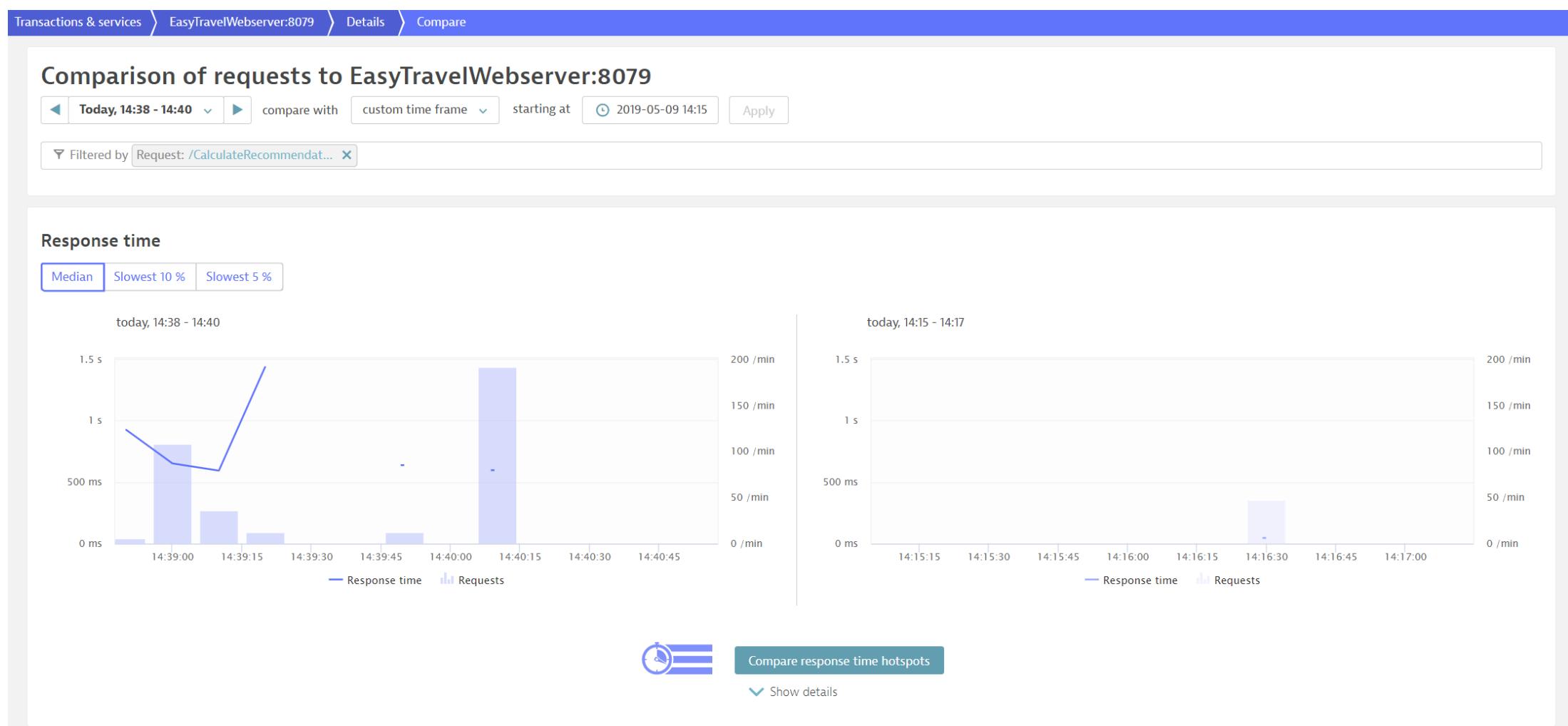
EasyTravelWebserver:8079

- Requests: 6 of 6
- Failed requests: 0 of 6
- Filter service flow to show only those transactions that contain the current call chain
- View PurePaths
- See every single request in PurePath view
- View web requests
- show more
- 2 Apache Tomcat services

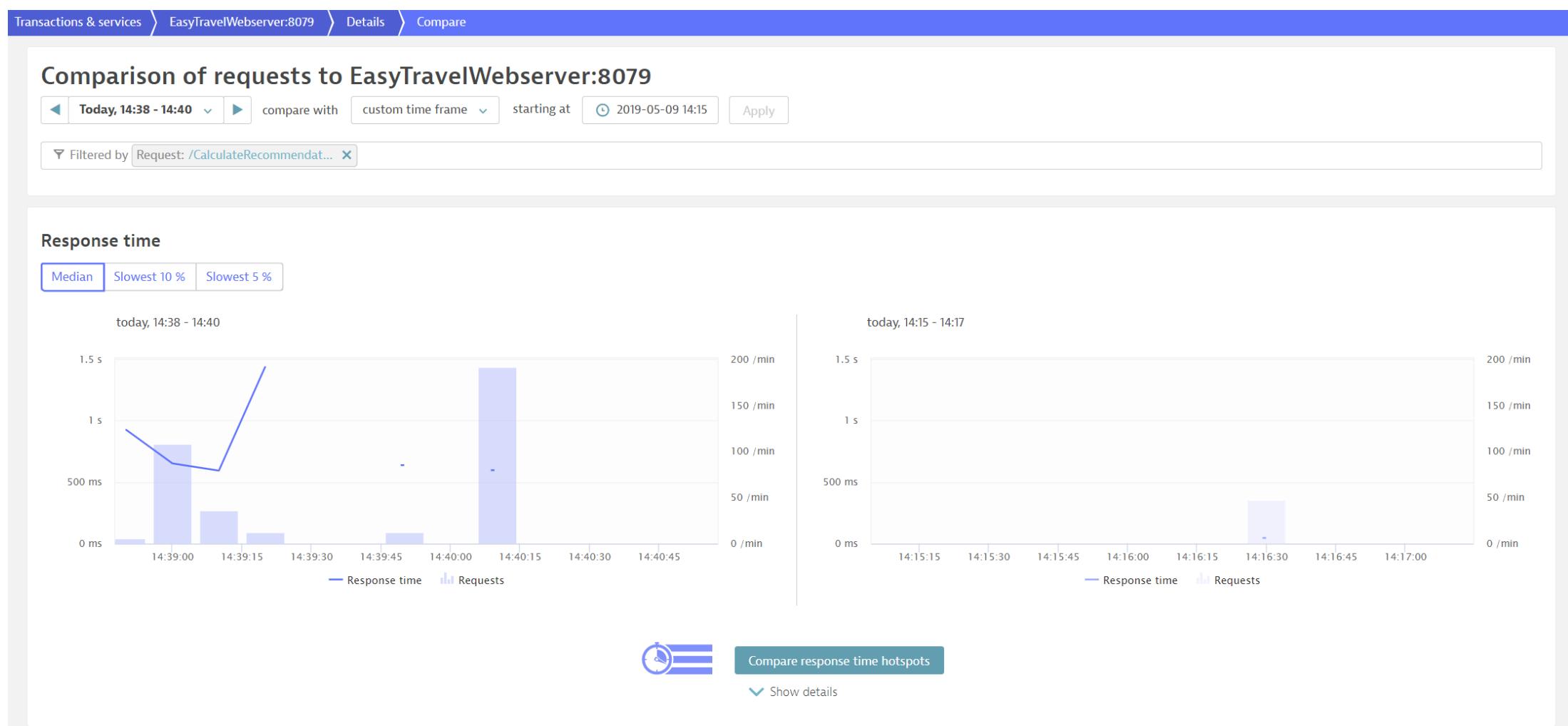
easyTravelBusiness

- Requests calling easyTravelBusiness via 2 other services
- Avg. response time: 21.4 ms
- Requests: 150
- Failed requests: 0
- View database statements
- Understand and analyze the most frequent and expensive database statements

Use Case #4 – Slow Loading Recommendations (cont.)



Use Case #4 – Slow Loading Recommendations (cont.)



Use Case #4 – Slow Loading Recommendations (cont.)

Transactions & services > EasyTravelWebserver:8079 > Details > Response time analysis

Average response time of requests of 'EasyTravelWebserver:8079'

Need help? Filter by Request: /CalculateRecommendat... ×

Today, 14:38 - 14:40 compare with custom time frame starting at 2019-05-09 14:15 Apply

Distribution

- Interaction with services and queues
- No database usage
- Service execution

Response time
+606 ms
645 ms

Top findings

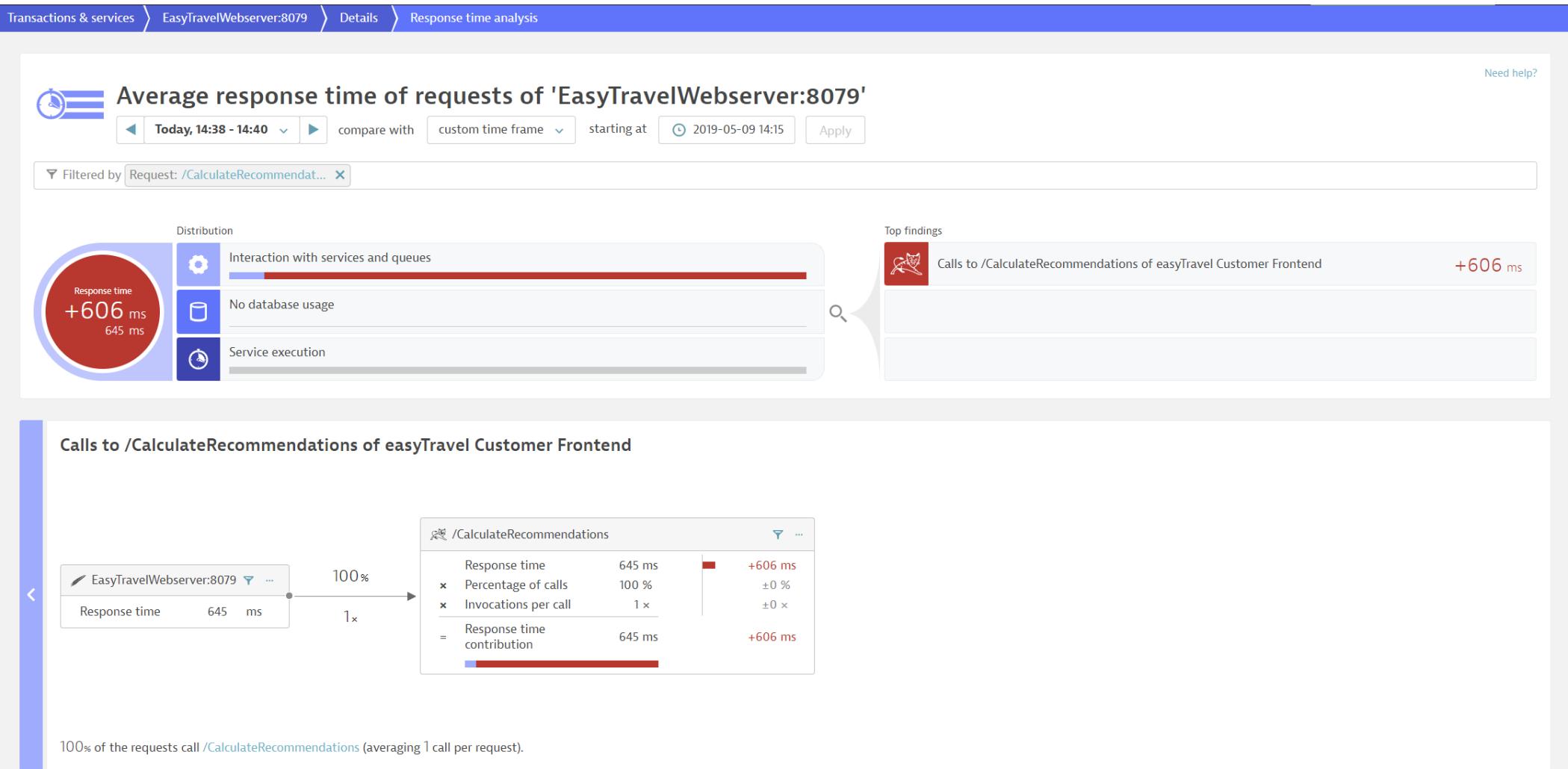
- Calls to /CalculateRecommendations of easyTravel Customer Frontend +606 ms

Calls to /CalculateRecommendations of easyTravel Customer Frontend

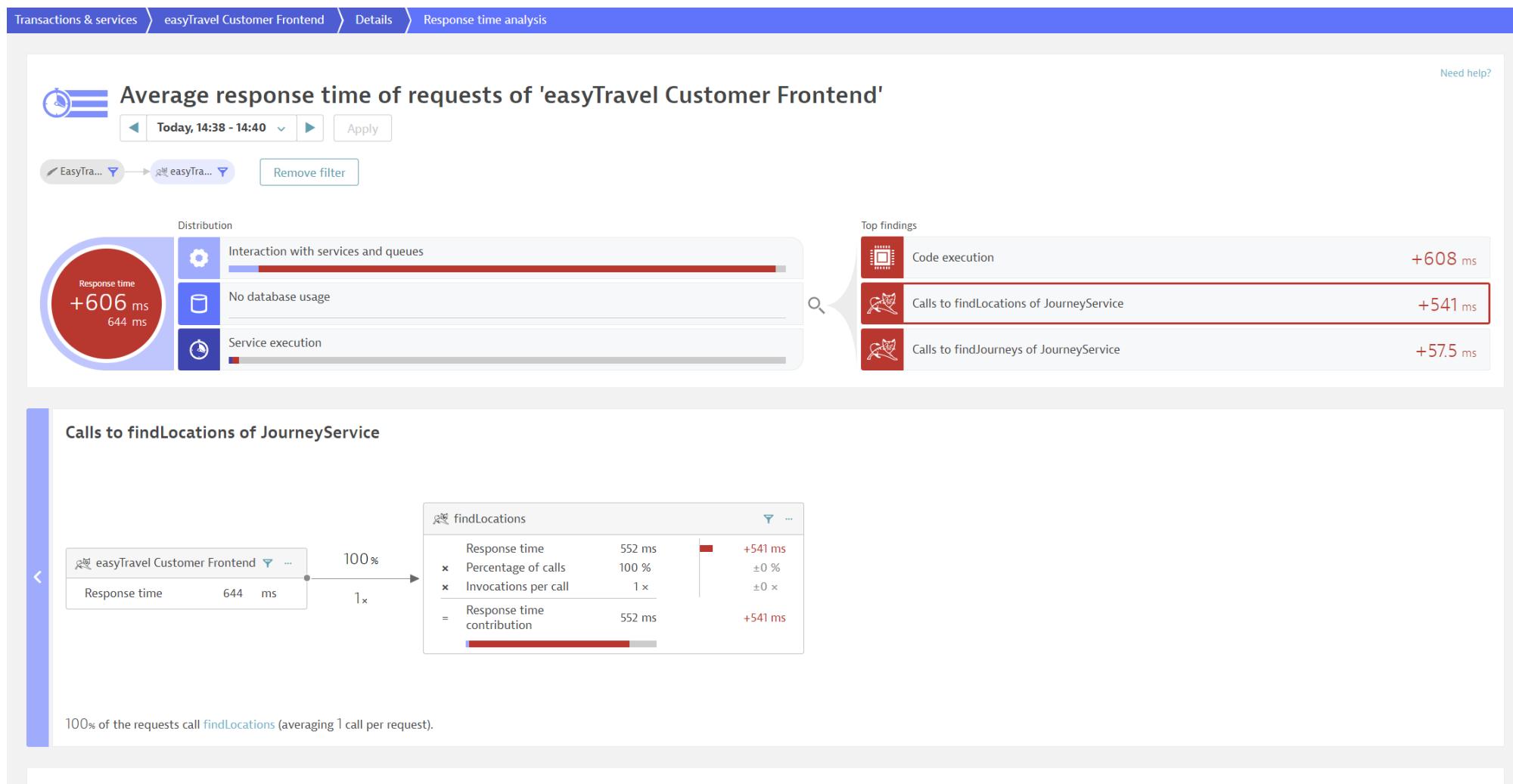
EasyTravelWebserver:8079 ... 100% →

/CalculateRecommendations	
Response time	645 ms
× Percentage of calls	100 %
× Invocations per call	1 x
= Response time contribution	+606 ms

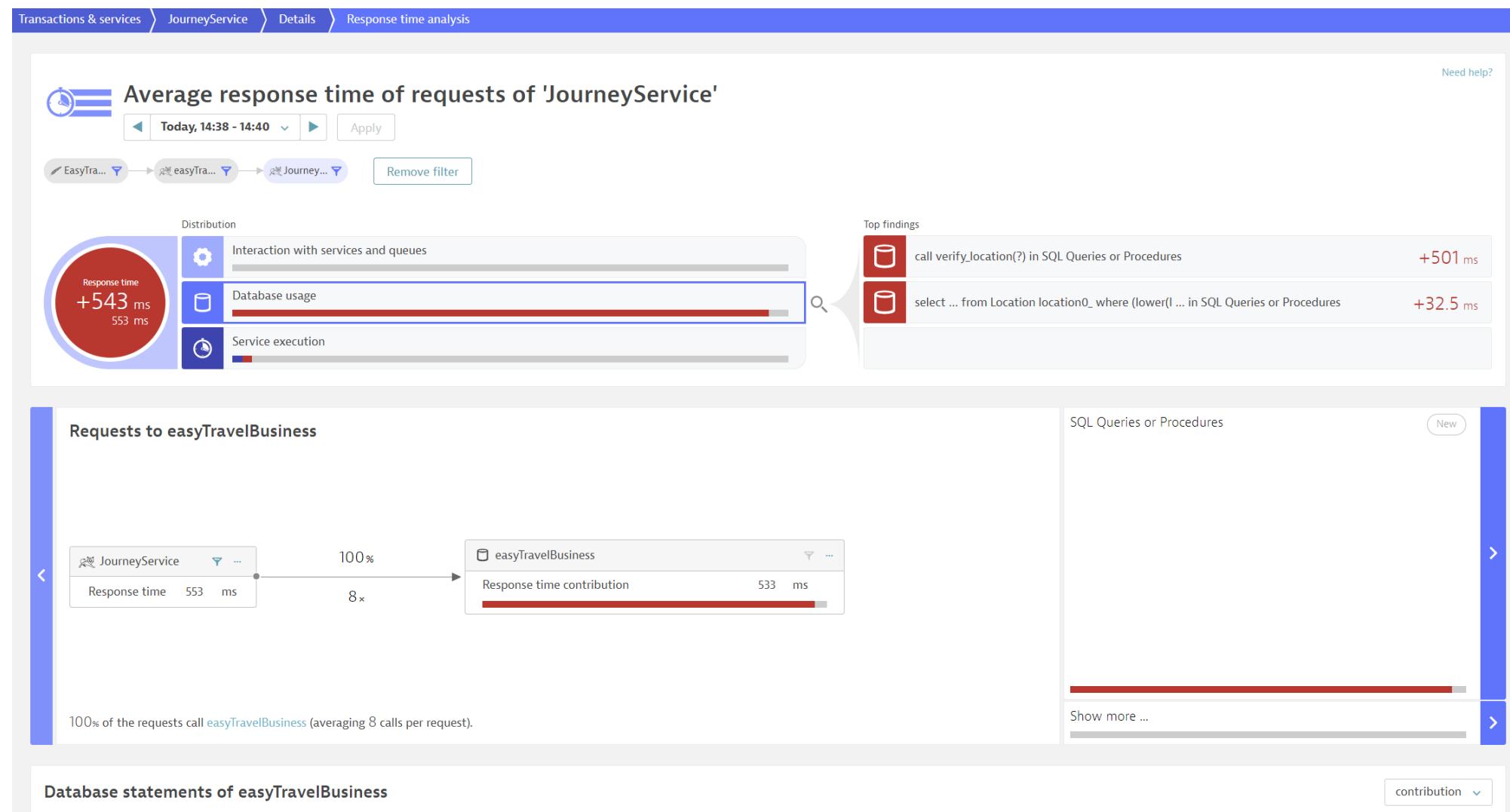
100% of the requests call /CalculateRecommendations (averaging 1 call per request).



Use Case #4 – Slow Loading Recommendations (cont.)



Use Case #4 – Slow Loading Recommendations (cont.)



Use Case #4 – Slow Loading Recommendations (cont.)

Transactions & services > JourneyService > Details > Response time analysis

Requests to easyTravelBusiness



100% of the requests call [easyTravelBusiness](#) (averaging 8 calls per request).

SQL Queries or Procedures

[New](#)

Show more ...

Database statements of easyTravelBusiness

New	call verify_location(?)	501 ms
x	Average execution time	501 ms
x	Percentage of calls	100 %
x	Invocations	1 x
=	Response time contribution	501 ms

+501 ms
+100 %
+1 x
+501 ms

[contribution](#) ▾

New	select location0_.name as name1_2_ from Location location0_ where (lower(location0_.name) like ('% ? %')) and (exists (select journey1_.id from Journey journey1_ where	32.5 ms
-----	---	---------

Use Case #4 – Slow Loading Recommendations (cont.)

Transactions & services > EasyTravelWebserver:8079 > Details > Backtrace

Service-level backtrace of requests of 'EasyTravelWebserver:8079'

Today, 14:43 - 14:45 | Apply

Filtered by Request: /CalculateRecommendat... X

The services and applications listed below make calls to this service. The tree view represents the sequence of services and application user actions that led to this service call, beginning with the page load or user action in the browser that triggered the sequence. Click to see which specific requests and user actions called this service.

Incoming requests to this service

- EasyTravelWebserver:8079
 - Apache Web Server easyTravel
- EasyTravel
 - Web application

48 Requests
0 Failed requests

18 User actions

User actions from EasyTravel

- Loading of page /orange.jsf
- click on "Search" on page /orange.jsf

10 User actions

8 User actions

Analysis of EasyTravel user actions



- At least 18 analyzed EasyTravel user actions trigger 48 calls to EasyTravelWebserver:8079
 - No EasyTravel user actions failed
 - No failed calls to EasyTravelWebserver:8079

Tips & Tricks

Returning HTTP 200 on Failure

- Try to avoid returning a HTTP 200 to the user, when the transaction actually fails:

The screenshot shows a Dynatrace Performance Monitor interface. On the left, a sidebar lists transactions: '/orange-booking-payment.jsf' (EasyTravelWebserver:8079), '/orange-booking-payment.jsf' (easyTravel Customer Frontend), 'checkCreditCard' (BookingService), and 'SocketNativeApplication.sendAndReceive' (Credit Card Validation). The main area displays the details for the transaction '/orange-booking-payment.jsf'. The 'Summary' tab is selected, showing the following metadata:

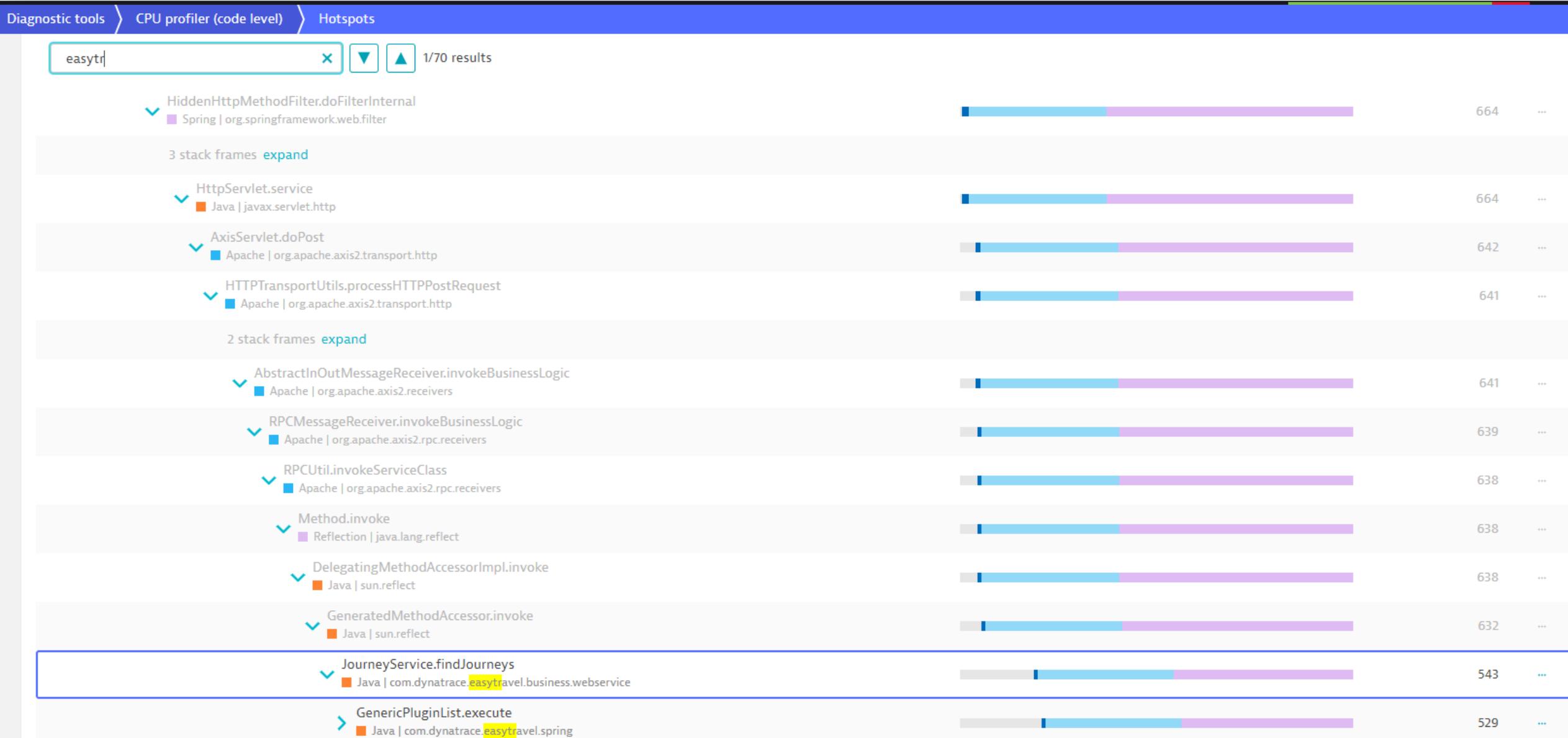
Service	EasyTravelWebserver:8079
Request	/orange-booking-payment.jsf
Host	ip-172-31-81-2.ec2.internal
Process	Apache Web Server easyTravel
Proxy	ip-172-31-81-2.ec2.internal (172.31.81.2)
Service type	Web request service
Service main technology	Apache HTTP Server
Technology	Apache HTTP Server (2.4.29)
Operating system	Linux (x86)
Bitness	64-bit

The 'Metadata' section shows the following details:

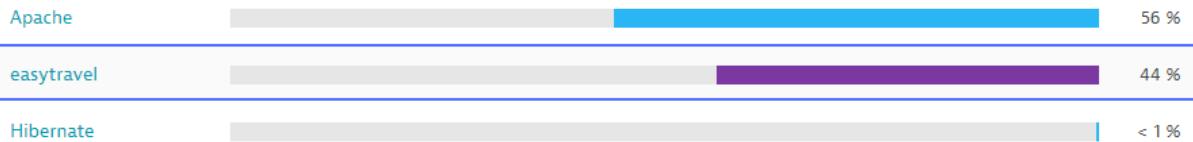
URI	/orange-booking-payment.jsf?journeyId=5461
HTTP method	POST
Response status	200 - OK
Context root	/

A large green rounded rectangle highlights the '200 - OK' status in the metadata table, with the text 'HTTP 200 Returned to user' overlaid.

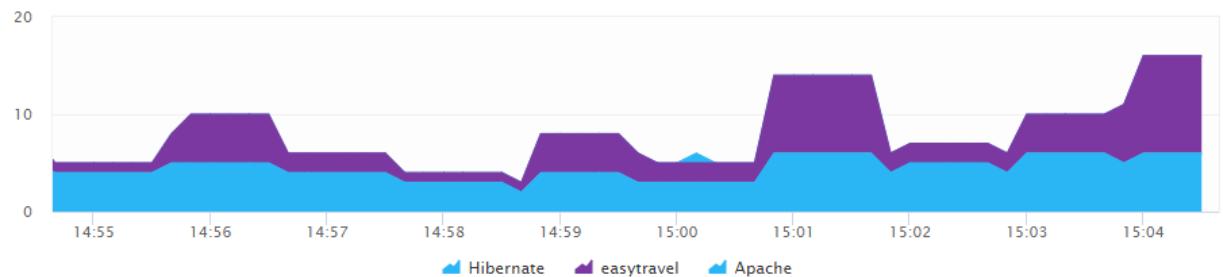
Finding your code in the CPU Profiler



Execution time breakdown



Top APIs

Filtered by API: easytravel X

Call tree

Hotspots

Top contributors

Method	Contribution	Stacktrace samples
JourneyUpdate.doExecute easytravel com.dynatrace.easytravel.database	200	200
SocketInputStream.socketRead0 Java java.net	77	77
CorrelationNative.registerCachedString Java <agent>	1	1

Calling methods of hotspot 'JourneyUpdate.doExecute'

Method	Contribution	Stacktrace samples	Actions
JourneyUpdate.doExecute easytravel com.dynatrace.easytravel.database	507	507	...
2 stack frames expand			
GenericPluginList.execute easytravel com.dynatrace.easytravel.spring	507	507	...
JourneyService.findJourneys easytravel com.dynatrace.easytravel.business.webservice	491	491	...

Properly Set up Tags

- Get Process Groups
 - Including tags
- Services CAN include tags
 - Service Name
- Process Groups
- Tagging rules

-qa

Last call 46 seconds ago

[Smartscape view](#) [...](#)

[Properties and tags](#)

[dev](#) **Tagged as Dev**

Detected name Tomcat/localhost
 Type Web request service
 Process group  -qa -dev **Process Group QA**
 Service main technology Apache Tomcat
 Process technology Apache Tomcat (8.5.28.0), Java (OpenJDK 1.8.0_181), Java (OpenJDK 1.8.0_191), and MUSLC
 Web server name Tomcat/localhost
 Context root /

 0 Applications
 0 Services
 1 Network client

26 /min Throughput

 3 Apache To...  3 Databases

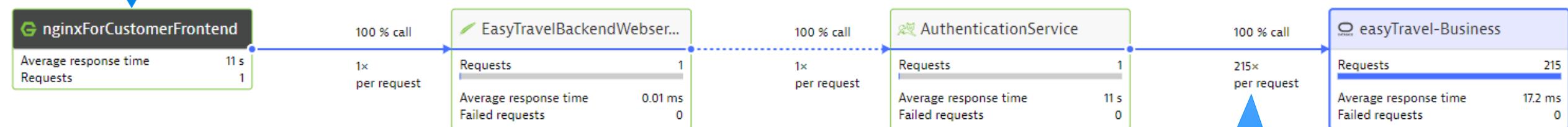
Processes and hosts

Process	Runs on	State
-qa	 Unmonitored 18 hours 21 minutes ago	 Available
-qa -dev	 Available	 Available
-qa -dev	 Available	 Available

QA and Dev mixed

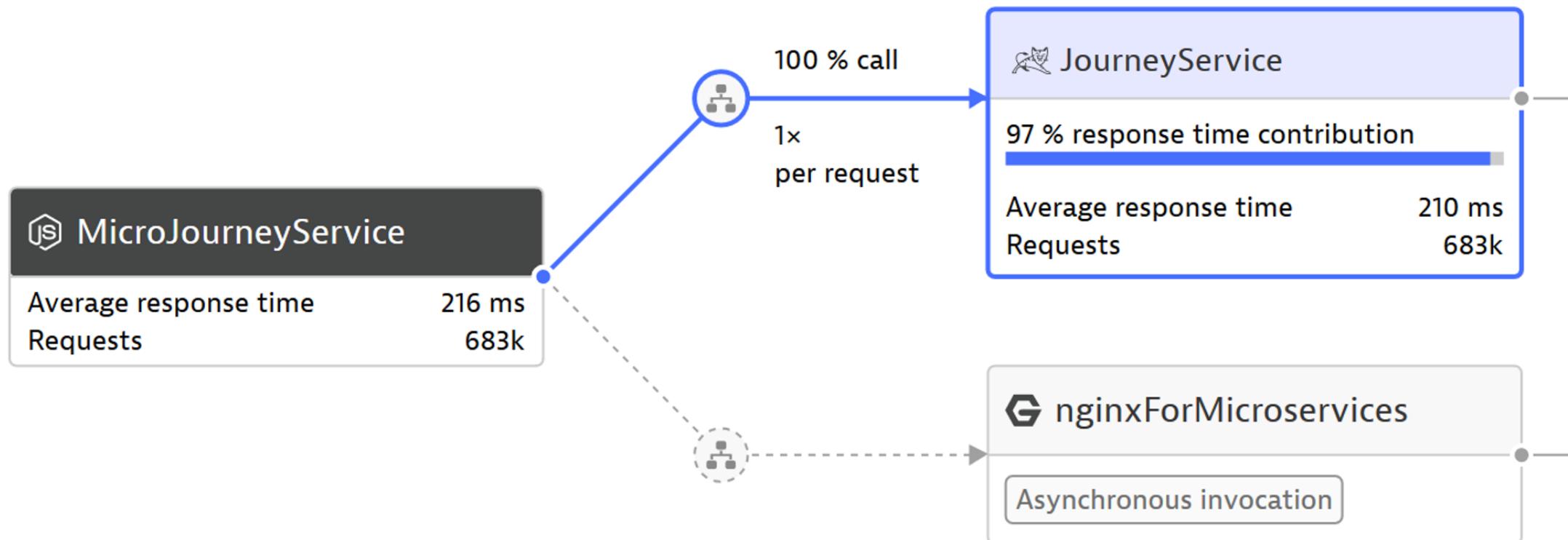
DB Spamming

Total Transaction
took 11s, for a
single request



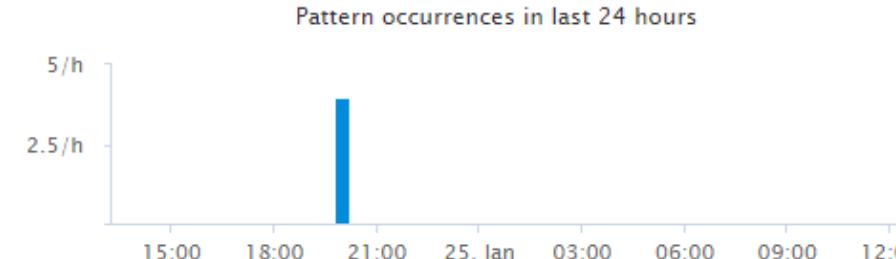
Single request
spawns 215 DB
queries!

Microservices – Considerations for tightly coupled services



Don't forget about the Logs

- Using Log Event Detection can help find application errors
 - Common Use Cases:
 - Detect Application Errors in Native Processes
 - Detect Errors with the Application Infrastructure (syslog, Windows Event Logs, etc...)
 - Detect Errors in CloudTrail logs

Event name	Details
OS Process Error	<p>Text pattern</p> <p>OS Process Error</p> <p>Detection condition</p> <p>Generates Log Error Problem if number of occurrences is higher than 0/min</p> <p>Detection scope</p> <p>CouchDB_ET 1 log on 1 host selected</p>  <p>Pattern occurrences in last 24 hours</p> <p>5/h</p> <p>2.5/h</p> <p>15:00 18:00 21:00 25. Jan 03:00 06:00 09:00 12:00</p> <p>Delete rule Edit rule</p>

Remember to Delete your AWS Resources!!!



Thank you

