# Spring Insight Operations

VMware vFabric Cloud Application Platform 5.0

Spring Insight Operations 1.5.1

VMware vFabric tc Server 2.6

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-000647-00

vmware

You can find the most up-to-date technical documentation on the VMware Web site at: https://www.vmware.com/support.

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com



VMware, Inc., 3401 Hillview Avenue, Palo Alto, CA 94304

www.vmware.com

# Table of Contents

# 1. About Spring Insight Operations

*Spring Insight Operations* describes Spring Insight concepts and provides instructions for configuring and using Spring Insight Operations with VMware® vFabric™ tc Server. Spring Insight Operations, part of VMware vFabric™ Spring Edition, provides in-depth performance monitoring of Web applications and tc Servers in production environments.

## Intended Audience

*Spring Insight Operations* is intended for operations teams and developers who use Spring Insight Operations to monitor and tune Web applications running on tc Server application servers in production environments.

# 2. Introduction to Spring Insight Operations

Spring Insight Operations gives you real-time visibility into web application and tc Server performance in production environments. Insight Operations graphs the health of applications over time for the entire cluster and for each server in the cluster. You see application and server problems as they occur, with detailed information about contributing events.

### Subtopics



## How Spring Insight Operations Works

Spring Insight Operations captures Web application events known as *traces*. A trace represents a thread of execution. It is usually started by an HTTP request but can also be started by a background job. A trace contains operations, which represent significant points in the execution of the trace, for example, a JDBC query or transaction commit. Insight Operations uses trace data to calculate summary information and to lead you to the specifics of why your application is not performing as well as it should be.

Insight Operations has a two-tiered architecture that consists of Spring Insight Dashboard and Spring Insight Agent. tc Runtime instances that run Web applications run the Spring Insight Agent application. Insight Agent collects trace data and sends it to a single, dedicated tc Runtime instance running Spring Insight Dashboard. Insight Dashboard enables you to monitor applications across all servers and to get detailed statistics from individual servers. The two-tiered architecture separates the collection of trace data by Insight Agent from the Insight Dashboard user interface, thereby ensuring that minimal overhead is added to tc Runtime instances that execute Web applications.

Spring Insight Dashboard and Spring Insight Agent are themselves Web applications, easily added to tc Runtime instances using tc Server templates.

## Difference Between Spring Insight Operations and Spring Insight Developer

Spring Insight Operations is designed to be deployed over a cluster of tc Runtime instances in production environments. Spring Insight Developer is designed for Web application development on a single tc Runtime instance during the development phase. The two Spring Insight products have a similar Dashboard user interface and collect similar information, except that Spring Insight Operations provides the additional capability to collect and view performance by server and overall for the cluster.

Because it runs on a single tc Runtime instance, Spring Insight Developer has a simpler architecture. Traces are collected and viewed on the same tc Runtime instance where the Web application is running. It is packaged as a template in vFabric tc Server Developer edition. With only one tc Runtime instance, Spring Insight Developer is trivial to set up. On the downside, it places greater resource demands on the tc Runtime instance than a tc Runtime instance running the Insight Agent component of Spring Insight Operations. That is, Spring Insight Developer is not designed for production deployments.

Both Spring Insight Developer and Spring Insight Operations can be customized by creating plug-ins using the Spring Insight Developer Kit. Custom plug-ins can collect traces from new execution points or define new ways to analyze and display trace information in the Dashboard. Plug-in development is documented in *Spring Insight Developer* documentation. The plug-ins you develop can be used with both Spring Insight Developer and Spring Insight Operations deployments.

## Use Cases for Spring Insight Developer and Spring Insight Operations

Spring Insight Developer and Spring Insight Operations have specific use cases throughout the application development and production cycle.

## Agile Development

While developing Web applications, using Spring Insight Developer allows the Web application developer to gain a substantial increase in productivity as they make changes and see the effect immediately. Typically, a developer makes changes to HTML or JSP and reloads the browser to verify that the modified application renders the Web page appropriately. However, developers often lack a centralized tool that shows how their changes affect the following factors:

- JDBC queries
- Spring bean interaction
- Calls to external services

Large, popular frameworks such as Hibernate and Spring Web push much of the code that developers formerly wrote manually into a convenient library. This process saves time and improves maintenance. The downside is relinquished control of code, which means that the developer might not know exactly what is going on behind the scenes:

- How many database transactions did a Web request create?
- How expensive is it to use complex Web parameter binding?
- What HTTP headers are sent to the developer's REST application?

The Spring Insight Trace answers these questions so that developers make changes and verify their effectiveness immediately.

Developers who are developing Web applications to be deployed on a cluster can use Spring Insight Operations to test, diagnose, and tune their applications running on a cluster. The cluster introduces interactions and performance factors that are not present when developing and testing on a single tc Runtime instance. Spring Insight Operations provides an opportunity for developers to quickly diagnose issues related to clustering and tune the application for a multi-server platform before deployment.

## QA Rear-View Mirror

Using Spring Insight Developer or Spring Insight Operations gives QA a richer picture of an application's performance and eliminates much of the work required to diagnose problems. As QA tests an application, problems arise:

- Slow-loading pages
- Database grinding
- Stack traces

As these problems occur, QA engineers can browse to the Spring Insight Dashboard and review all recent operations. Engineers can access in-depth information that helps them track down bugs:

- Database queries and their performance
- Detailed description of the web request, its parameters, and headers
- Component method calls and their parameters

- All Spring components and their performance

QA forwards this information to the developer and improves the turnaround time for identifying and resolving root causes.

## Testing Load and Performance

Spring Insight Operations works with your existing load-testing tools to observe how your application behaves under load on a cluster before deploying to a production setting. You can use Spring Insight Operations to:

- View graphs of overall application health on the cluster and on individual servers
- View relative throughput and response times among servers in the cluster as load varies
- Detect any application performance degradation or failure during long-running tests
- Drill down into a trace on a specific server to see what exactly happened under heavy load
- Discover resource contention or bottlenecks

Repeated load and performance testing with Spring Insight Operations helps you to diagnose performance issues and to tune and harden your clustered application.

## Extended Customization with the Spring Insight Developer Kit

Spring Insight uses a plug-in architecture to collect, correlate, and analyze performance metrics from your application and from different pieces of third-party software and libraries that interact with the application. With the Spring Insight Development Kit., you can create plug-ins that are tailored to the needs of your application and that extend Insight functionality:

- Collect trace details such as Hibernate flush times and JSP render times.
- Render custom HTML for an operation detail frame.
- Analyze traces to produce new types of Endpoints, such as a JMS message queue.

The Developer Kit is a separate download. It includes a copy of the source code for all plug-ins shipped with Spring Insight and a sample plug-in that serves as a starting point for creating your own plug-ins.

For details about the Development Kit, see *Spring Insight Developer*.

## Browser Performance Data from Integration with Google SpeedTracer

SpeedTracer is a Google Chrome extension that analyzes how your application is performing inside the browser. It measures how long the browser takes to render, transform CSS, show images, and so on. VMware partners with Google to integrate Spring Insight data into SpeedTracer and tie browser performance to backend performance with a robust client-server application performance tool. If your Web application uses Ajax and other rich open Web technologies, try SpeedTracer with Spring Insight.

For details, see *Spring Insight Developer*.

## Production Application Monitoring and Troubleshooting

When Web applications move to production, Spring Insight Operations helps administrators monitor and troubleshoot the health of the applications and the servers running them. Spring Insight Operations collects trace information from multiple servers and presents information for the entire cluster.

When you start Spring Insight Operations Dashboard, you can see a graph of application health for the entire cluster. Explore the applications by clicking a single application or by expanding an application and clicking on its endpoints. You see any problems immediately, including which particular applications and endpoints are problematic.

Spring Insight Operations has a Servers section beneath the Applications section that lets you display health and detail information for each server in the cluster. You can quickly identify servers that experience errors and that underperform. You can

see trace details for specific events and exceptions, so that when problems occur you have easily accessible, detailed information to help you pinpoint the cause.

## Supported Environments

Spring Insight Operations runs in environments with a Java 6 JVM. Spring Insight supports the same configurations as tc Runtime. See the supported configurations information in the *vFabric tc Server Release Notes*.

# 3. Installing and Configuring Spring Insight Operations

To install Spring Insight Operations, you set up at least two tc Runtime instances. A single dedicated instance runs Spring Insight Dashboard. One or more other tc Runtime instances execute Web applications and are configured with the Spring Insight Agent template. The Spring Insight Agent software on each non-Dashboard server submits application trace information to the Dashboard server, where the information is compiled and displayed in the Dashboard user interface.

If you have obtained tc Server as part of the VMware vFabric Cloud Application Platform, complete the license activation and installation procedures in *Getting Started with vFabric Cloud Application Platform* before you continue with these installation instructions.

### Subtopics



## Download tc Server and Spring Insight Operations

Spring Insight Operations is distributed in two bundles, one for the Spring Insight Dashboard, and one for the Spring Insight Agent. If you have not already installed tc Server, you must also download and install vFabric tc Server Standard Edition. You can download ZIP or compressed TAR files, depending on your operating system and preference.

- vFabric tc Server Standard Edition, for example `vfabric-tc-server-standard-edition-2.6.0.RELEASE.tar.gz`
- Spring Insight Dashboard Template, for example `insight-vfabric-tc-server-dashboard-1.6.0.RELEASE.tar.gz`
- Spring Insight Agent Template, for example `insight-vfabric-tc-server-agent-1.6.0.RELEASE.tar.gz`

**Procedure**

1. Go to the VMware download page at http://downloads.vmware.com and navigate to the vFabric tc Server page.
2. Download the tc Server Standard Edition bundle.
3. Download the Spring Insight Dashboard and Spring Insight Agent bundles.

## Install and Start Spring Insight Dashboard

The Spring Insight Dashboard web application runs on one tc Runtime instance. You connect to the Insight Dashboard server to view the Spring Insight user interface. .

**Prerequisites**

Download the software as described in Download tc Server and Spring Insight Operations.

**Procedure**

1. On the computer that will run the Spring Insight Dashboard, create a directory for tc Server.

   For example: **`/home/tcserver`**

2. Copy or upload the tc Server Standard Edition ZIP or compressed TAR file you downloaded into the directory that you created for tc Server.

3. Extract the tc Server bundle into the tc Server directory.

   The top-level directory created when you extract the bundle is the tc Server installation directory.

   For example: **`/home/tcserver/tc-server-standard-edition-2.6.0.RELEASE`**

   This directory contains the tc Runtime utility scripts, the `templates` directory, tc Runtime 6 and 7, and so on.

4. Copy or upload the Spring Insight Dashboard ZIP or compressed TAR file you downloaded into the tc Server directory.

5. Extract the Spring Insight Dashboard bundle into the `templates` subdirectory of the tc Server installation directory.

   This action creates a directory with the name `insight-dashboard` in the `templates` subdirectory.

6. Create the tc Server instance that will run Insight Dashboard.

   a. In a Unix terminal window or Windows Command Prompt, change to the tc Server installation directory.

   b. Use the `tcruntime-instance.sh` or `tcruntime-instance.bat` command to create the tc Server instance that will run Insight Dashboard.

      For example, to create an instance named myDashboard, enter the following command without line breaks:

      ```
      prompt$ ./tcruntime-instance.sh create myDashboard -t insight-dashboard
         -p insight-dashboard.dashboard.jms.bind.uri=tcp://myDashboardServer:21234
      ```

   The table describes the `tcruntime-instance` command options in the example.

   *Table 3.1. Main Options of the tcruntime-instance Command*

| Command Option | Description |
|---|---|
| **`-t`** | Specifies the template to use to create the instance. It is the name of the `insight-dashboard` directory you extracted into the `templates` directory. |
| **`-p`** | Sets the property `insight-dashboard.dashboard.jms.bind.uri` to the URI that Spring Insight Dashboard uses to communicate with the Spring Insight Agent instances. The prefix `insight-dashboard` is the name of the Dashboard template in the `templates` directory. The bind URI has three parts: a protocol (`tcp` in the example); host name or IP number (`myDashboardServer` in the example); and port number (`21234`). You supply an identical URI when you create Insight Agent instances. |
| **`-v`** | (Optional) Specifies the version of tc Runtime to use for this instance. By default, `tcruntime-instance` uses the highest tc Runtime version present in the tc Server installation directory. If you want to use a lower version, you must supply the `-v` option. |

7. Ensure that the `dashboard.jms.bin.uri` port specified in the previous step (`21234` by default) is open in the firewall to permit Spring Insight Agent instances to connect.

8. Start the Spring Insight Dashboard instance with the `tcruntime-ctl.sh` or `tcruntime-ctl.bat` command.

- On Unix:

```
prompt$ ./tcruntime-ctl.sh myDashboard start
```

- On a Windows system, you must first install the instance as a Windows Service and then start it. You have to install the instance only once.

```
prompt> tcruntime-ctl.bat myDashboard install
prompt> tcruntime-ctl.bat myDashboard start
```

**What to do next**

- Install Spring Insight Agent with each tc Server instance that will run your Web application. See Install and Start Spring Insight Agent.
- For detailed information about additional tc Runtime command options and creating tc Server instances, see the vFabric tc Server documentation.

## Install and Start Spring Insight Agent

The Spring Insight Agent software running on each tc Server instance submits application trace information to the server that hosts Spring Insight Dashboard, where the information is compiled and displayed in the Insight Dashboard user interface.

Perform the steps in this procedure for each tc Server instance that will participate in your Web application cluster, except for the server that hosts Insight Dashboard.

**Prerequisites**

- Download the software as described in Download tc Server and Spring Insight Operations.
- Install Spring Insight Dashboard as described in Install and Start Spring Insight Dashboard.

**Procedure**

1. On each computer where you will install a tc Server instance, except for the server that runs Spring Insight Dashboard, create a directory for tc Server.

   For example: **`/home/tcserver`**.

2. Copy or upload the tc Server Standard Edition ZIP or compressed TAR file you downloaded into the directory that you created for tc Server.
3. Extract the tc Server Standard Edition ZIP or compressed TAR file into the directory you created.

   The top-level directory that is created when you extract the bundle is the tc Server installation directory.

   For example: **`/home/tcserver/tc-server-standard-edition-2.6.0.RELEASE`**.

   This directory contains the tc Runtime utility scripts, the `templates` directory, tc Runtime 6 and 7, and so on.

4. Copy or upload the Spring Insight Agent ZIP or compressed TAR file you downloaded into the tc Server directory.
5. Extract the Spring Insight Agent Template bundle into the `templates` subdirectory of the tc Server installation directory.

   This action creates a directory with the name `insight-agent` in the `templates` subdirectory.

6. In a Unix terminal window or Windows command prompt, change to the tc Server installation directory and use the `tcruntime-instance.sh` or `tcruntime-instance.bat` command to create the tc Server instance that will run Spring Insight Agent.

For example, enter the following command on one line:

```
prompt$ ./tcruntime-instance.sh create MyInstanceN -t insight-agent
    -p insight-agent.dashboard.jms.connect.uri=tcp://myDashboardServer:21234
```

The table describes the `tcruntime-instance` command options in the example.

*Table 3.2. Main Options of the tcruntime-instance Command*

| Command Option | Description |
|---|---|
| **`-t`** | Specifies the template to use to create the instance. It is the name of the `insight-agent` directory you extracted into the `templates` directory. |
| **`-p`** | Sets the property `insight-agent.dashboard.jms.connect.uri` to the URI that Insight Agent uses to communicate with Insight Dashboard. Replace *`myDashboardServer`* with the name or IP number of the computer running the Dashboard instance. The port number must match the port number specified when you installed the Spring Insight Dashboard instance (`21234`, by default). |

7. Start the tc Server instance.

   On Unix:

   ```
   prompt$ ./tcruntime-ctl.sh myInstanceN start
   ```

   On Windows, you must install the instance as a Windows service before you start it. Perform this command one time only:

   ```
   prompt> tcruntime-ctl.bat myInstanceN install
   prompt> tcruntime-ctl.bat myInstanceN start
   ```

8. Deploy your Web applications to the tc Server instance.

   Hot-deploy an application by copying its WAR file into the *`myInstanceN`*`/webapps` directory.

**What to do next**

- Set configuration properties for Spring Insight Agent. See Configure Spring Insight Operations.
- For information about all `tcruntime-instance` options, see the `tcruntime-instance` command reference in Getting Started with tc Server.

## Connect to the Spring Insight Dashboard Server

You can connect to the server that is running Insight Dashboard from any computer in the network.

**Procedure**

1. Use a web browser to access Spring Insight Dashboard at `http://`*`myDashboardServer`*`:8080/insight`.
2. Log in with a username-password combination listed in the following table.

   *Table 3.3. Insight Dashboard Login Information*

| Role | User Name | Password |
|---|---|---|
| insight_admin | `admin` | `insight` |
| insight | `spring` | `insight` |

   Users with the insight_admin role have access to administrative functions and support tools in the Dashboard.

**What to do next**

- Change the username and password or create new ones in the `CATALINA_HOME/conf/tomcat-users.xml` file of the tc Server instance that is running Insight Dashboard. See Configure Spring Insight Dashboard Server Users.
- If you use an LDAP server, you can configure Spring Insight Dashboard to authenticate users against the LDAP server. See (Optional) Configuring Spring Insight Dashboard to Authenticate with LDAP.

# Configure Spring Insight Dashboard Server Users

The Spring Insight Operations Dashboard by default uses the `CATALINA_HOME/conf/tomcat.users` file to authenticate users. Users can be regular users (`insight` role) or administrators (`insight and insight_admin` roles). The `insight_admin` role is required to perform some operations on the Administration page.

You should secure your Spring Insight Operations installation by changing the default users and passwords.

## Procedure

1. Edit the `CATALINA_HOME/conf/tomcat.users` file. This is the initial content of the file:

```
<?xml version="1.0"?>
<tomcat-users>
    <role rolename="insight"/>
    <role rolename="insight_admin"/>
    <user password="insight"
          roles="insight"
          username="spring"/>
    <user password="insight"
          roles="insight,insight_admin"
          username="admin"/>
</tomcat-users>
```

2. Edit or add `<user>` elements for the users you wish to authenticate, setting the `password`, `roles`, and `username` attributes for each user. Include the `insight` role for all users and add the `insight_admin` role for users who should have administrator capabilities.
3. After saving changes to `tomcat.users`, restart the Spring Insight Operations Dashboard, using the `tcruntime-ctl` restart command.

## What to do next

- If you have an LDAP server, you can change the Dashboard's authentication realm to use it for authenticating users. See Configure the Spring Insight Dashboard to Authenticate with LDAP.
- To specify the username and password that Agents use to connect with the Dashboard, modify the `CATALINA_HOST/insight/insight.properties` files in the Dashboard and every Agent tc Runtime instance. See Configure Spring Insight Operations for instructions.

# Configure Spring Insight Operations

You configure Spring Insight Operations by editing the `insight.properties` files for each Spring Insight Dashboard and Spring Insight Agent instance. The `insight.properties` file is a standard Java properties file. Each property is on its own line. Properties have the form *`property: value`*. Comments begin with #. The properties differ for Spring Insight Dashboard and Spring Insight Agent.

**Prerequisites**

- Connect to the server that is running Spring Insight Dashboard. See Connect to the Spring Insight Dashboard.
- Review the tables in Properties File for Spring Insight Dashboard and Properties File for Spring Insight Agent. You will use this information when you configure `insight.properties.`

**Procedure**

1. On the computer that is running Spring Insight Dashboard, edit the `insight.properties` file. Refer to the table in Properties File for Spring Insight Dashboard for information about valid property names and values.

   For example: `/home/tcserver/tc-server-standard-edition/myInstance/insight/insight.properties`

   Restart the tc Runtime instance after saving your edits.

2. On a computer that is running Spring Insight Agent, edit the `insight.properties` file. Refer to the table in Properties File for Spring Insight Agent for information about valid property names and values.

   For example: `/home/tcserver/tc-server-standard-edition/myInstance/insight/insight.properties`

   Restart the tc Runtime instance after saving your edits.

3. Repeat step 2 to edit the `insight.properties` file for each tc Server instance that is running Spring Insight Agent.

## Properties File for Spring Insight Dashboard

You set properties for an Insight Dashboard instance in the `insight.properties` file for that instance.

Regarding the database properties in the table, Spring Insight bundles the H2 Java SQL database and uses it internally to persist its data. A tc Runtime instance configured with Spring Insight is automatically configured to connect to and use the H2 database.

*Table 3.4. Properties of the Spring Insight Dashboard insight.properties File*

| Property | Description |
|---|---|
| `insight.data.dir` | Full pathname of the directory in which Spring Insight persists its data. Default: *`INSTANCE_DIR`*`/insight/data`. |
| `dashboard.jms.data.dir` | Directory in which JMS data is persisted. This property appears only in the `insight.properties` files for Insight Dashboard. |
| `dashboard.jms.bind.uri` | Protocol, host name or IP number, and port number for the Insight Dashboard instance to which Insight Agent instances connect and send application trace data. This property appears only in the `insight.properties` files for Insight Dashboard. The property corresponds to the `dashboard.jms.connect.uri` property in the Insight Agent `insight.properties` file.<br><br>You specify the value for this parameter in the format *`protocol`*`://`*`host`*`:`*`port`*.<br><br>• *`protocol`*`:tcp`, `nio`, or `ssl`.<br><br>• *`host`*: Computer name or IP number of the host where the Dashboard instance is executing.<br><br>• *`port`*: Port number for the connection. The default port is 21234. |
| `agent.auth.`*`login`* | Login name and password an Agent instance uses to log in to a Dashboard instance. The *`login`* portion of the property is the login name. The value is the password. The default, which specifies the login name `agent` and password `insight` is:<br><br>`agent.auth.agent: insight`<br><br>Customizing login names does not modify the names of agents connecting to Insight Dashboard, only their ability to connect to the Dashboard. The ability to modify settings is useful if you are managing several deployment environments through a single Dashboard instance and want to selectively enable or disable connections from sets of agents, based on the login they use. |
| `application.context. ignore.`*`identifier`* | (Optional) Application contexts that you want Spring Insight to ignore. Spring Insight does not instrument these applications, which improves the startup time of the application. |

| Property | Description |
|---|---|
| | The default `insight.properties` file excludes the Spring Insight application itself from being instrumented by adding the following entry:<br><br>application.context.ignore.dashboard: localhost\|insight<br><br>The *identifier* part of the property name (such as `dashboard` in the example above) is not used internally by Spring Insight but must be unique within the properties file. Use this part of the property name to separate multiple entries. Specify an identifier that has meaning to you.<br><br>The value of this property must be of the form `localhost\|`*`context`* where *`context`* refers to the URL context that invokes the application. The value of *context* refers to the URL context you specify in a browser to invoke the application. By default, this is the name of the WAR file in which the application is packaged, but you can specify a different name in the `web.xml` file.<br><br>The following example ignores the Tomcat Manager application, which has the application context `manager`.<br><br>`application.context.ignore.mgr: localhost\|manager` |
| `database.url` | URL of the database that Spring Insight uses to persist its data. This value defaults to the H2 Java SQL database that is bundled with Spring Insight. |
| `database.driverClassName` | Class name of the JDBC Driver that Spring Insight uses to connect to the database. Default: `org.h2.Driver`. |
| `database.username` | Name of the user who connects to the database that Spring Insight uses to persist its data. Default: `insight`. |
| `database.password` | Password of the user who connects to the database that Spring Insight uses to persist its data. Default: `admin`. |
| `trace.exclude.path.`*`type`* | List of patterns that are matched against requests. When a trace matches a path in this list, the trace is discarded. This setting lets you ignore requests to static content such as `.css` or `.js` files. Specify this property multiple times, one for each *type* of pattern you want to exclude.<br><br>For example:<br><br>**trace.exclude.path.png: /**/*.png**<br>**trace.exclude.path.gif: /**/*.gif**<br>**trace.exclude.path.jpg: /**/*.jpg**<br>**trace.exclude.path.css: /**/*.css**<br>**trace.exclude.path.js: /**/*.js** |

## Properties File for Spring Insight Agent

You set properties for a Spring Insight Agent instance in the `insight.properties` file for that Instance.

*Table 3.5. Properties of the Spring Insight Agent insight.properties File*

| Property | Description |
|---|---|
| `dashboard.jms.connect.uri` | Protocol, host name or IP number, and port number of the Insight Dashboard instance to which Insight Agent instances connect and send application trace data. The property appears only in the `insight.properties` files for Agents and the value must match the `dashboard.jms.bind.uri` property in the Dashboard `insight.properties` file.<br><br>You specify the value for this property in the format *`protocol`*`://`*`host`*`:`*`port`*.<br><br>• *`protocol`*`:` `tcp`, `nio`, or `ssl`.<br><br>• *`host`*`:` Host name or IP number of the computer running the Dashboard instance.<br><br>• *`port`*`:` Available port number for the connection. Default: `21234`. |

| Property | Description |
|---|---|
| `agent.auth` and `agent.auth.password` | Credentials used to connect to Insight Dashboard.<br><br>The value of `agent.auth` must match the *login* value of the `agent.auth.`*`login`* property in the Insight Dashboard `insight.properties` file. Default: `agent` .<br>The `agent.auth.password` is the value specified by the Insight Dashboard `agent.auth.`*`login`* property. Default: `insight`. |
| `application.name.override` | Name that overrides the default name for a deployed application in data sent to Insight Dashboard. This override can be useful when you deploy different applications with the same name on different Insight Agent instances. Multiple overrides are allowed. Overrides take the form `application.name.override.`*`hostName`*`\|`*`contextName`*`:` *`hostName`*`\|`*`contextName`*.<br><br>The initial *`hostName`*`\|`*`contextName`* specifies the default application host and application context name. The second *`hostName`*`\|`*`contextName`* are the host name and context name to substitute in data forwarded to Insight Dashboard.<br><br>Strip out forward slashes from context names. |
| `agent.heartbeat.interval` | Frequency, in seconds, with which the Insight Agent instance sends a heartbeat to Insight Dashboard. Default: `15`. |
| `agent.trace.buffer` | Maximum number of traces to store in memory before analysis. Default: `20000`. |
| `agent.traces.rate.initial` | Initial maximum number of traces to send per minute to Insight Dashboard. Default: `200`. |
| `agent.name.override` | Unique fixed ID for the Insight Agent instance. By default, the name is constructed from the hostname and pid of the Agent's JVM process. Therefore, Insight Dashboard treats each newly started Insight Agent instance as a new Agent, to accommodate rapidly changing application server clusters. If you prefer, you can set the value of this property to a fixed name for the Agent. |
| `trace.filter.similar.per.hour` | Rate per hour that similar traces are filtered. If a trace has the same end point and tag signature as another recent trace, it is not sent to Insight Dashboard. (Only one trace with a given signature will be sent every 5 minutes.) A value of 4 filters traces so that only one trace with a given signature is sent every 15 minutes. Default: `12`. |
| `application.context.ignore` | Application context that the Insight Agent should not instrument. Ignoring applications improves the startup time for the application. No traces for these applications are sent to Insight Dashboard. Multiple entries are allowed.<br><br>The format for this property is `application.context.ignore.`*`name`*`:` *`hostName`*`\|`*`contextName`*`.`<br><br>*`name`*: Value that makes the property name unique in the properties file; for example, the context name for the application you want to ignore.<br><br>*`hostName`* and *`contextName`*: Host name of the application host and the application context name.<br><br>The Insight Agent application is automatically added, to prevent instrumenting and sending traces from the Agent application:<br><br>`application.context.ignore.agent: localhost\|insight-agent` |

# (Optional) Configure the Spring Insight Dashboard to Authenticate with LDAP

If you use LDAP, you can use it to authenticate users logging in to the Insight Dashboard. The examples shown in the procedure are for OpenLDAP.

**Prerequisites**

Install Spring Insight Dashboard and create a tc Runtime instance. See Install and Start Spring Insight Dashboard .

**Procedure**

1. Copy the `insight-dashboard-security-ldap-1.5.x.RELEASE.jar` and `insight-plugin-dashboard-security-ldap.xml` files from the `extras/ldap` directory to the `insight/dashboard-plugins` directory of your Dashboard tc Runtime instance.

   For example, enter the following commands, each on a single line without line breaks:

   ```
   prompt$ cd /home/tcserver/vfabric-tc-server-standard-edition-2.6.0.RELEASE/myDashboard
   prompt$ cp extras/ldap/* insight/dashboard-plugins
   ```

2. Remove the default `security-tcserver.jar.`

   ```
   prompt$ rm insight/dashboard-plugins/insight-dashboard-security-tcserver-1.1.x.RELEASE.jar
   ```

   The `dashboard-plugins` directory may contain either `security-tcserve.jar` or `security-ldap.jar`, but not both.

3. `In insight/dashboard-plugins/plugin-config/ insight-plugin-dashboard-security-ldap.xml`, modify bean `contextSource` to point to your own LDAP server.

   ```
   <bean id="contextSource"
         class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
     <constructor-arg value="ldap://myOpenLDAPServer:389/dc=springsource,dc=com"/>
     <property name="userDn" value="cn=Manager,dc=springsource,dc=com"/>
     <property name="password" value="secret"/>
   </bean>
   ```

4. Add `userDN` and `password` properties that the Directory Manager or other user will use to connect to the LDAP server.

   If the LDAP server allows anonymous access you do not need to add `userDN` and `password` properties.

   ```
   <bean id="contextSource"
         class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
     <constructor-arg value="ldap://myOpenLDAPServer:389/dc=springsource,dc=com"/>
     <property name="userDn" value="cn=Manager,dc=springsource,dc=com"/>
     <property name="password" value="secret"/>
   </bean>
   ```

5. In the `insightAuthenticationProvider` bean, modify the `userDnPatterns` property to specify your User Base DN.

   ```
   <property name="userDnPatterns">
            <list><value>cn={0},ou=Users,o=SpringInsight</value></list>
   </property>
   ```

6. Modify the first `<constructor-arg>` element of the `TransformingLdapAuthoritiesPopulator` bean to point to your Group Base DN.

   ```
   <bean class="org.springframework.security.ldap.userdetails.DefaultLdapAuthoritiesPopulator">
     <constructor-arg ref="contextSource"/>
     <constructor-arg value="ou=Groups,o=SpringInsight"/>
     <property name="groupSearchFilter" value="(|(member={0})(member={1}))"/>
   </bean>
   ```

7. Set up the two Insight roles by modifying the `map` element of the second `<constructor-arg>` element to specify the groups for regular Insight users (Operators) and Insight administrators (Administrators).

```
<map>
  <entry key="Operators" value="insight"/>
  <entry key="Administrators" value="insight_admin"/>
</map>
```

# (Optional) Configure Spring Insight for SSL

You can configure Spring Insight Operations so that Insight Agents connect to Insight Dashboard securely through SSL. The Insight Agent and Insight Dashboard complete a handshake that includes establishing trust and negotiating data encryption parameters. The data that they exchange for the duration of the connection is encrypted and secured from eavesdropping or tampering.

SSL uses public key encryption, which requires creating certificates (keys), keystores, and truststores. These procedures use the `keytool` utility included with the Sun JDK to create self-signed certificates and stores. Self-signed means that the certificates you generate are not signed by a Certificate Authority, such as VeriSign or Thawte.

1. Set Up SSL for Spring Insight Dashboard.

   Configure Spring Insight Dashboard so that Insight Agents can connect securely through SSL.

2. Set up SSL for Spring Insight Agent.

   Perform these steps for each tc Server instance that is running Spring Insight Agent.

## Set Up SSL for Spring Insight Dashboard

If you already have keystores, you can use them in this procedure instead of generating new ones. This procedure uses the `keytool` utility included with the Sun JDK to create self-signed certificates and stores. If you prefer to use a CA-signed certificate, purchase one from a CA such as VeriSign or Thawte. For help creating a Certificate Signing Request (CSR) and importing the signed certificate and trusted certificates into your keystore, see the documentation for keytool.

You can use another SSL toolset, such as OpenSSL. See the documentation for your toolset for the correct commands to use in the following procedures.

**Procedure**

1. Change to the directory where you want to create the keystore, for example *`CATALINA_BASE`*`/conf`.
2. If you do not already have a keystore file, create one with the following command:

   ```
   prompt$ keytool -genkey -alias dashboard -keyalg RSA -keystore dashboard.keystore
   ```

3. Enter the requested information at the prompts.

   This information is encoded into the certificate the command creates. Make a note of the key password for use in later commands.

   The command creates the file `dashboard.keystore` containing one entry with the alias `dashboard`.

4. Export the dashboard certificate.

   ```
   prompt$ keytool -export -alias dashboard -keystore dashboard.keystore -file dashboard_cert
   ```

5. At the prompt, enter the keystore password.

   The file `dashboard_cert` is created in the current directory. You will need this file later when you create truststores for the Agents.

6. Edit *`CATALINA_BASE`*`/insight/insight.properties` and change the `dashboard.jms.bind.uri` property to use the SSL scheme.

```
dashboard.jms.bind.uri: ssl://localhost:20234
```

7. Set the Java system properties to specify the location of the `dashboard.keystore` file and the keystore password.

   It is easiest to do this in the *`CATALINA_BASE`*`/bin/setenv.sh` script. Edit the `setenv.sh` script and add these lines above the `JAVA_OPTS=...` line.

```
SSL_KEYSTORE="/full/path/to/dashboard.keystore" # e.g. "$CATALINA_BASE/conf/dashboard.keystore"
SSL_KEYSTORE_PW="keystore_password"
SSL_OPTS="-Djavax.net.ssl.keystore=$SSL_KEYSTORE -Djavax.net.ssl.keystorePassword=$SSL_KEYSTORE_PW"
```

8. Add the `SSL_OPTS` environment variable to the `JAVA_OPTS` variable.

```
JAVA_OPTS="$JVM_OPTS $AGENT_PATHS $JAVA_AGENTS $JAVA_LIBARARY_PATH $SSL_OPTS"
```

To enable the changes, restart the tc Runtime instance.

## Set Up SSL for Spring Insight Agents

Set up SSL for Insight Agents so that they can connect securely to Insight Dashboard. Complete this procedure for each tc Server instance that is running Spring Insight Agent.

**Prerequisites**

- Complete the steps in Set Up SSL for Spring Insight Dashboard.
- Copy the `dashboard_cert` certificate file you exported in the previous procedure onto each Agent host computer, or make it available on a network drive or a thumb drive.

**Procedure**

1. Change to the directory where you want to create the Insight Agent keystore and truststore files, for example, *`CATALINA_BASE`*`/conf`.
2. Create a keystore for the agent, if one does not already exist.

```
keytool -genkey -alias agent -keyalg RSA -keystore agent.keystore
```

3. When prompted, enter the requested information.

   The identification information is encoded into the Agent's certificate. Make a note of the keystore password you create for later steps. The command creates the `agent.keystore` file in the current directory.

4. Create a truststore and import the dashboard public certificate.

```
keytool -import -alias dashboard -keystore agent.truststore -f /full/path/to/dashboard_cert
```

5. Edit *`CATALINA_BASE`*`/insight/insight.properties` and change the `dashboard.jms.connect.uri` property to use the SSL scheme.

```
dashboard.jms.connect.uri: ssl://dashboardHost:20234
```

6. Set the Java system properties to specify the location of the `agent.keystore` file, the location of the `agent.truststore` file, and the keystore password.

   It is easiest to do this in the *`CATALINA_BASE`*`/bin/setenv.sh` script. Edit the `setenv.sh` script and add the following lines before the `JAVA_OPTS=...` line.

```
SSL_KEYSTORE="/full/path/to/agent.keystore" # e.g. "$CATALINA_BASE/conf/agent.keystore"
```

```
SSL_TRUSTSTORE="/full/path/to/agent.trustore" # e.g. "$CATALINA_BASE/conf/agent.truststore"
SSL_KEYSTORE_PW="keystore_password"
SSL_OPTS="-Djavax.net.ssl.keystore=$SSL_KEYSTORE -Djavax.net.ssl.truststore=$SSL_TRUSTSTORE \
    -Djavax.net.ssl.keystorePassword=$SSL_KEYSTORE_PW"
```

7. Add the `SSL_OPTS` environment variable to the `JAVA_OPTS` variable.

```
JAVA_OPTS="$JVM_OPTS $AGENT_PATHS $JAVA_AGENTS $JAVA_LIBARARY_PATH $SSL_OPTS"
```

To enable these changes, restart the tc Runtime instance.

## (Optional) Increase tc Runtime Memory

Spring Insight might require increased memory for tc Runtime instances. You might have to increase the maximum heap or maximum PermGen size (two JVM options), or both.

**Procedure**

- Edit the *CATALINA_BASE*`/bin/setenv.sh|bat` file. *CATALINA_BASE* refers to the root directory of your instance, such as `/home/tcserver/vfabric-tc-server-standard-edition/insight-instance`. The comments in the file itself show examples of setting the JVM options.

# 4. Using Spring Insight Operations

Spring Insight Operations enables you to see at a glance how your applications and servers are doing. You can immediately see the overall health of the cluster and, with a few clicks, view the performance of individual applications, end points, or servers.

The following procedures describe basic tasks that help you start using Spring Insight Operations:

- Invoking Spring Insight in your Browser
- Browsing all Applications and Endpoints
- Viewing Recent Activity of Your Application
- Filtering Trace Details
- Importing and Exporting Traces
- Customizing Endpoint Thresholds
- Viewing Spring Insight Data In Google Speed Tracer

## Before You Begin

Before you start using Spring Insight Operations, set up your production application cluster and deploy applications. Setting up a clustering infrastructure is a complex topic and there are many alternatives available for load-balancing, managing servers, and deploying applications to a cluster. See Enabling Clustering for High Availability for information on this topic. The important considerations for using Spring Insight Operations with your cluster are:

- Install a dedicated Insight Operations Dashboard – create a dedicated tc Runtime instance for the Dashboard using the Spring Insight Operations Dashboard template.
- Install the Insight Operations Agent web application in your tc Runtime instances – use the Spring Insight Operations Agent template when you create the tc Runtime instances that run applications in your cluster. The Agent is configured to send application trace data to the Dashboard.

See Installing and Configuring Spring Insight Operations for detailed instructions for these tasks. For complete documentation about tc Server, see Getting Started with vFabric tc Server.

## Invoking Spring Insight Operations in your Browser

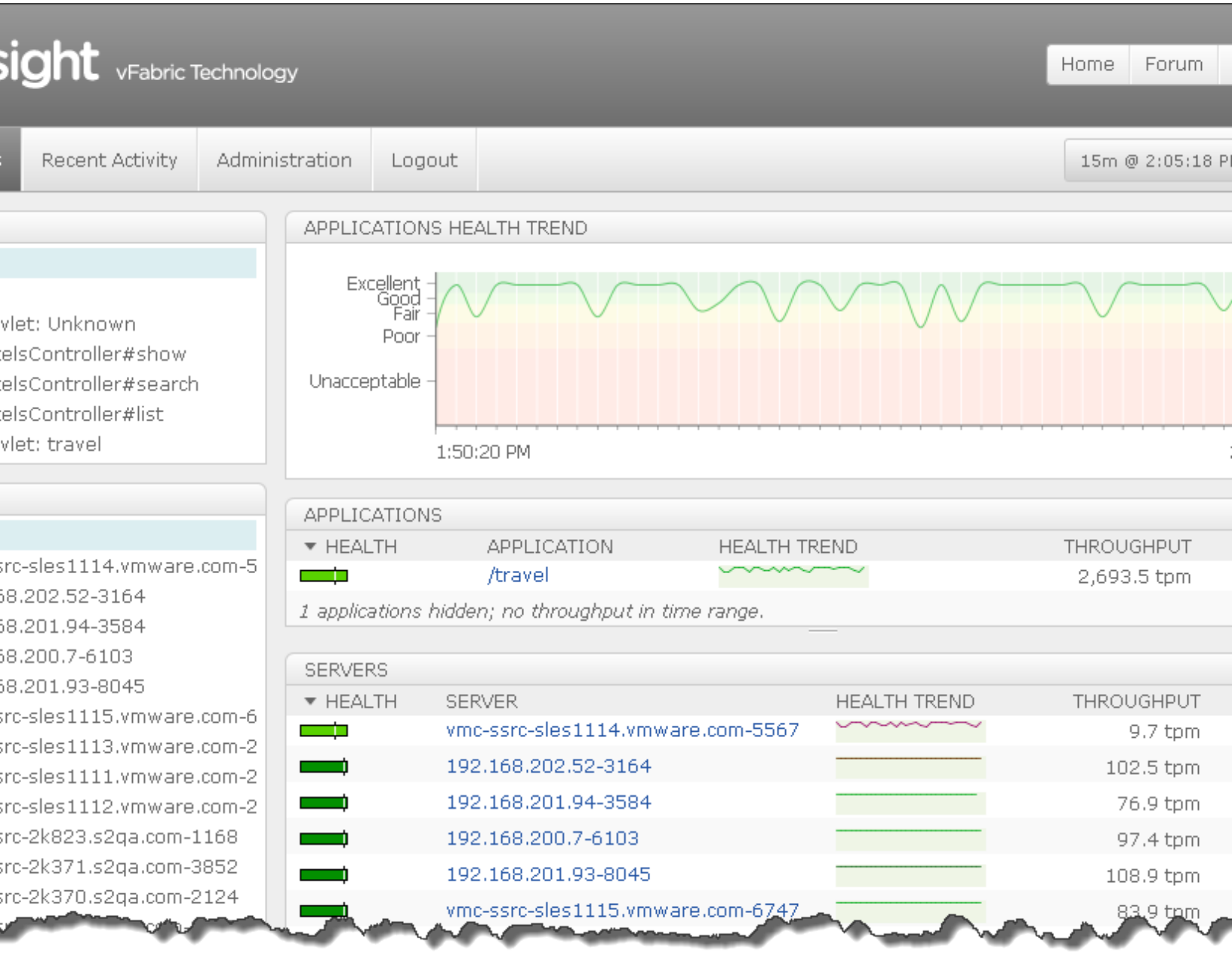
Follow these steps to invoke Spring Insight Operations in your browser and to start seeing trace data.

1. Use your browser to browse to the Spring Insight dashboard using the following URL:

   `http://dashboardServer:port/insight`

   Replace *`dashboardServer`* with the name or IP number of the host running the Dashboard and *`port`* with the HTTP listen port, 8080 by default. If you have set up SSL, you can also use `https://dashboardServer:SSLport/insight`. Replace *`SSLport`* with the configured SSL port number, 8443 by default.

   The browser invokes the main Spring Insight dashboard on the Browse Resources screen:

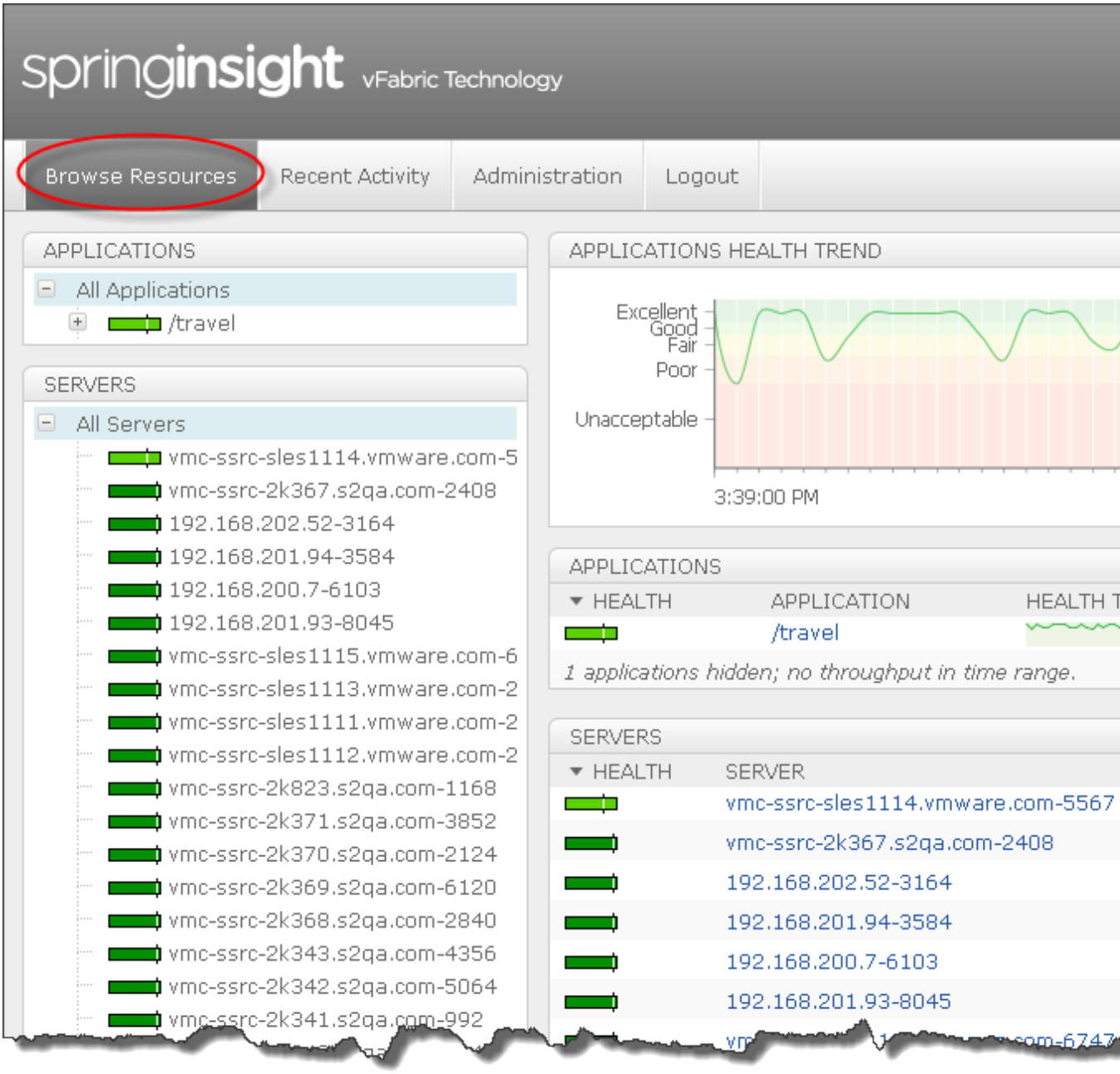



Spring Insight traces all application activity and displays it on the dashboard. For example, if you performed any JDBC queries, each one is shown in the Spring Insight dashboard along with a time line of recent requests.

## Browsing all Applications, Servers, and End Points

The Browse Resources screen shows information for all types of applications, but it is especially effective (and shows the most detail) for Spring, Grails, and Roo applications.

1. From the main Spring Insight dashboard, click the **Browse Resources** link:



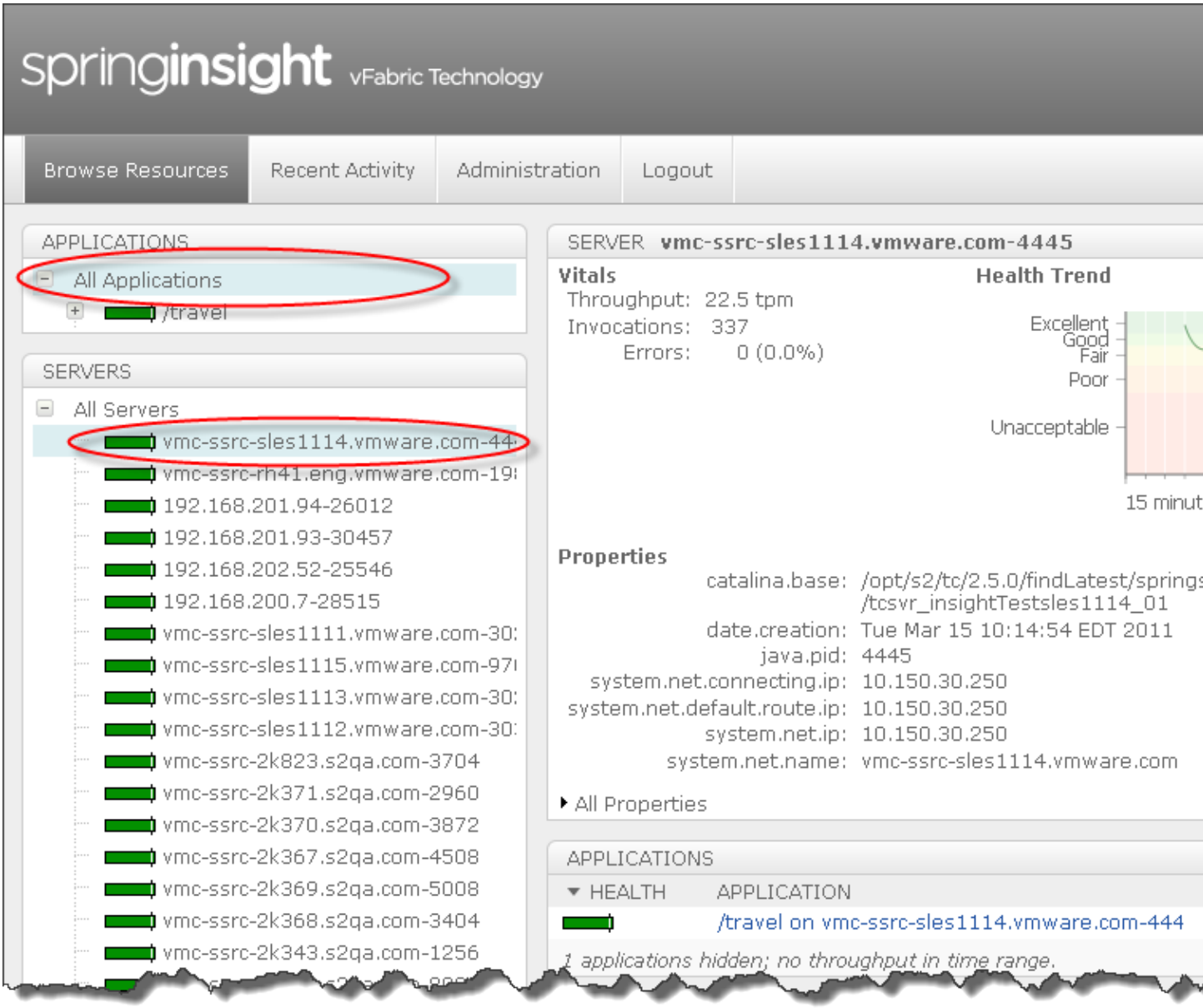

On the left, the root tree node, **All Applications**, is selected in the **APPLICATIONS** panel and **All Servers** is selected in the **SERVERS** panel. The **APPLICATIONS HEALTH TREND** panel displays a graph of the recent health of *all* applications on *all* servers. The **APPLICATIONS** panel on the right, beneath the graph, displays a list of the applications with some statistics for the application for the time period displayed by the graph:

- The **Health Trend** column shows a simple sparkline that graphically describes the recent health of the application.
- The **Throughput** column shows how many Traces per Minute (tpm) were executed over the current time window.
- The **Errors** column shows what percentage of Traces resulted in an Error (HTTP status 500 to 600).

The graphical or tabular view of all applications is useful to see which applications have been busiest and to compare their relative health. Click in the graphs to see the application-specific information. Click the column headers in the table to sort the information based on the values in the column.

The **SERVERS** panel at the bottom of the right column displays the same statistics for each server in the cluster. From this list, you can easily identify underperforming or failing servers.

2. With **All Applications** selected in the left **APPLICATIONS** panel, click on an individual server in the left **SERVERS** panel. This view displays a Health Trend graph for all applications running on the selected server and detailed information for the server.




The heading of the graph panel changes to identify the server you selected. The **Vitals** section of the graph panel displays performance statistics for the server for the time period depicted on the graph:

- **Throughput**: how many Traces per Minute (tpm) were executed during the current time window.

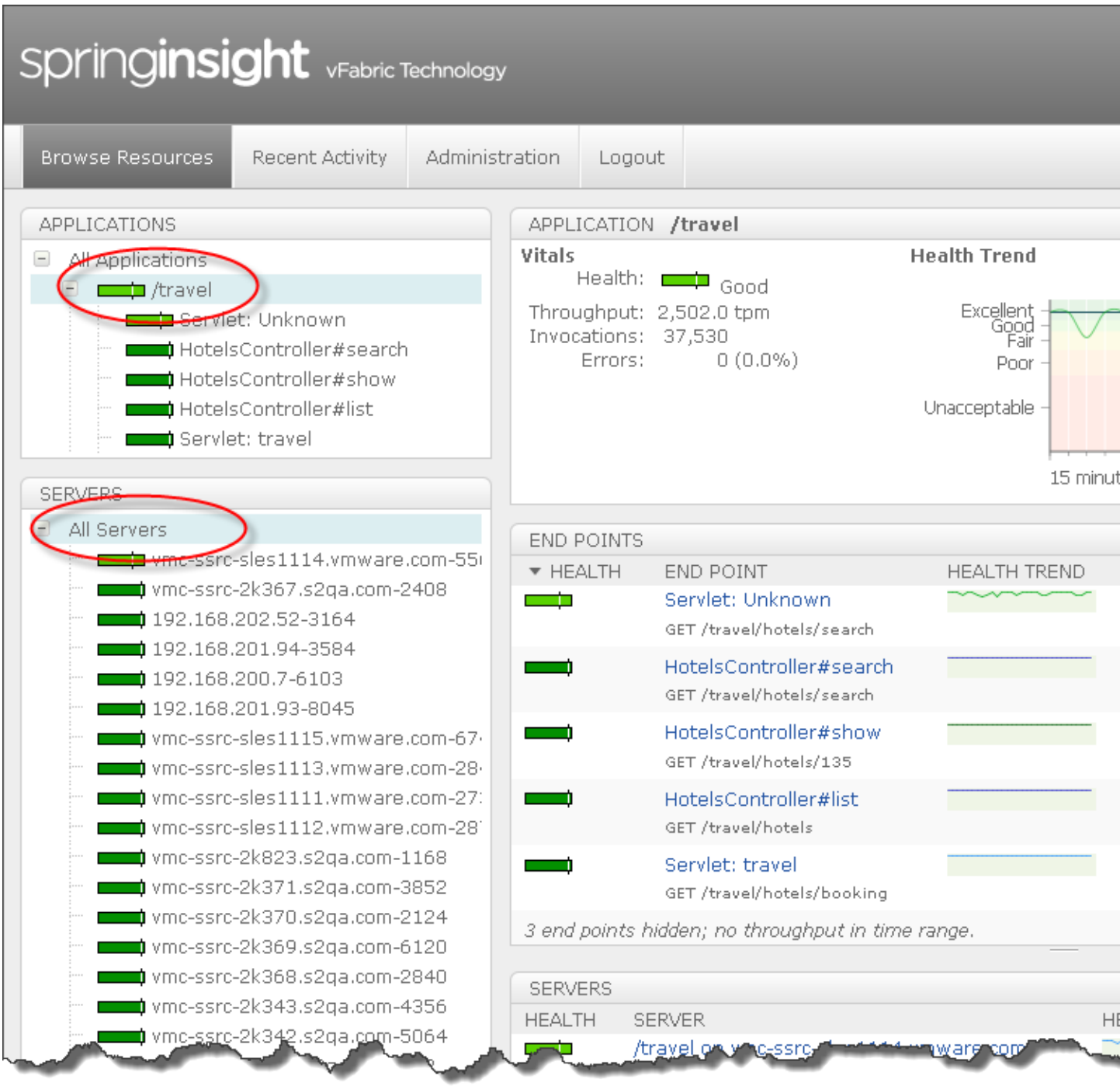
- **Invocations**: the total number of requests made during the current time window.
- **Error rate**: the percentage of invocations that resulted in an HTTP error status.

The **Properties** section of the graph panel lists system property settings for the server:

- **catalina.base**: the home directory for the tc Runtime instance.
- **date.creation**: the date the tc Runtime instance was created.
- **java.pid**: the process ID of the Java VM.
- **system.net.connecting.ip**: *needs description*
- **system.net.default.route.ip**: *needs description*
- **system.net.ip**: *needs description*
- **system.net.name**: The network name of the host computer.

Expand the **All Properties** list at the bottom to see a complete list of tc Server and Java system properties.

3. Click on an application in the left **APPLICATIONS** panel and **All Servers** in the left **SERVERS** panel. The right panel displays information about the selected application and its End Points across all servers in the cluster. The **END POINTS** panel displays the End Points associated with the application; the same list appears below the name of the application in the left **APPLICATIONS** panel.




The **Vitals** section in the graph shows a summary of the health of the application.

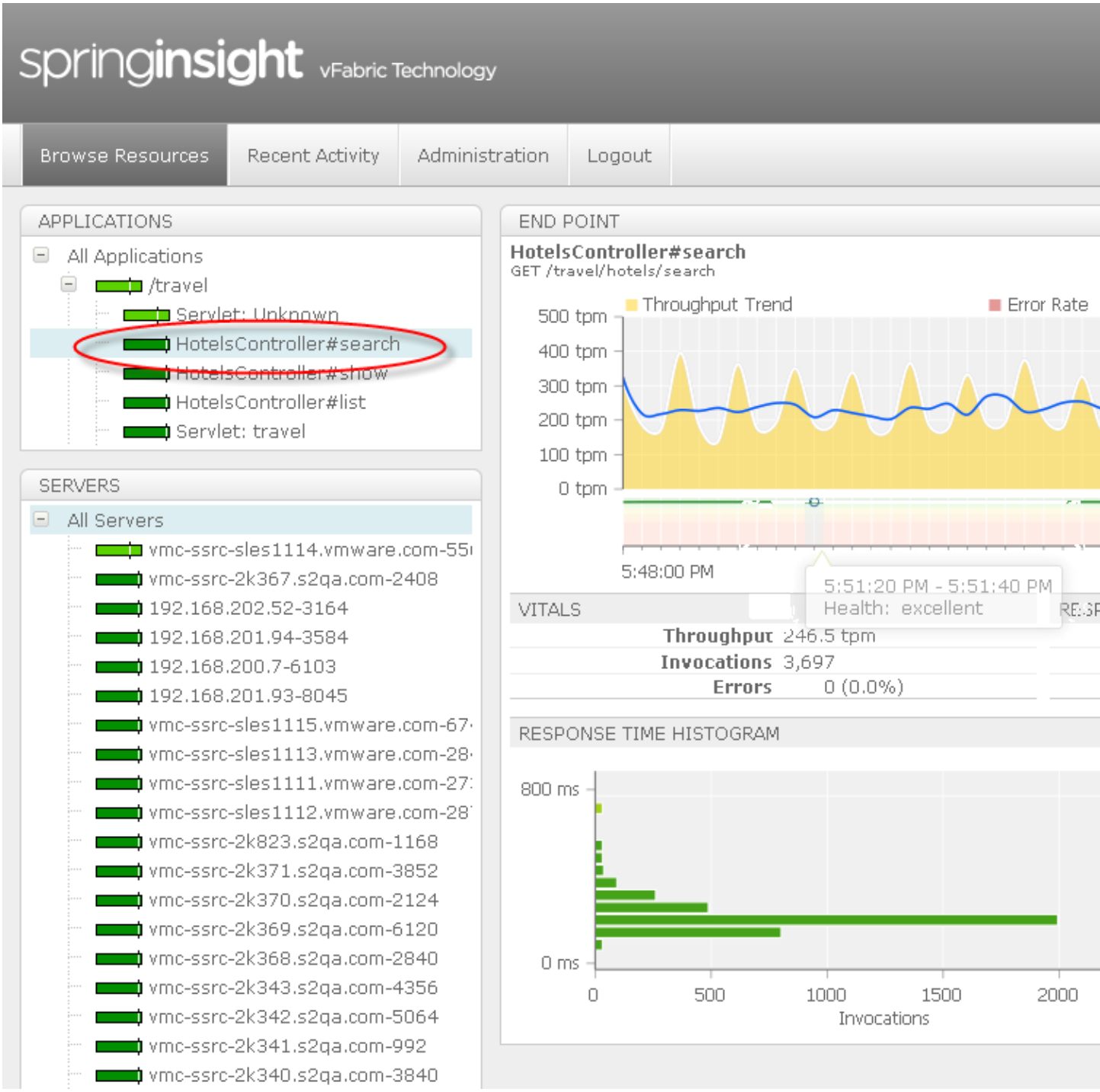
Each row in the **End Points** table represents an End Point, which is a receptor for requests. The universe of all possible HTTP URLs is unlimited. However, Spring Insight can group requests together based on the controller with which the requests are associated. For each End Point, Spring Insight displays the following information:

- **Health**: Shows how well the response time metric is kept within a tolerable threshold, where red is less healthy and green is more healthy. See Customizing End Point Thresholds for setting the tolerable threshold.

- **End Point**: Displays the name of the End Point.
- **Throughput Trend**: Displays a simple sparkline that shows the recent mean throughput time of the End Point.
- **Throughput**: Shows how many Traces per Minute (tpm) were executed over the current time window.
- **Errors**: Shows what percentage of Traces resulted in an Error (HTTP status 500 to 600).
- **Response Time** : Displays the 99% response time over the given time range. This value is most useful to determine the worst-case request. A value of 115 ms indicates that 99% of the requests completed within 115 milliseconds. The response time of an HTTP request is determined by the full time it takes the container to ship the response to the client, and not just the time spent in a controller.

4. Click on any individual server in the left **SERVERS** panel to see the same information described in the previous step for just the selected server.

5. Click on a particular End Point, either in the left **APPLICATIONS** panel or in the right **END POINTS** table for a particular application:




Spring Insight displays the following detailed information about the End Point:

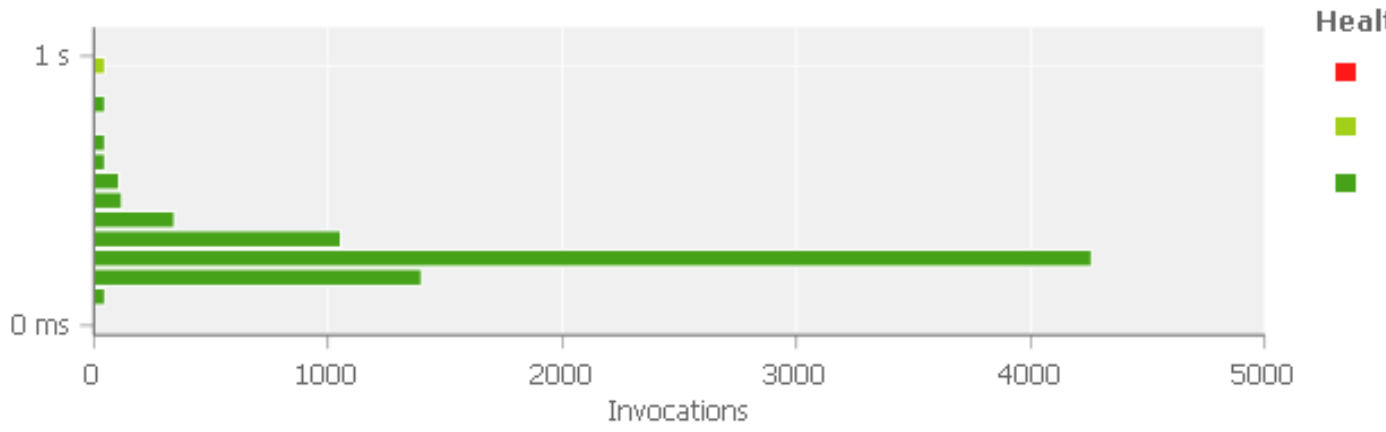
- The chart shows the throughput, response time, and error rate trends on the same graph. Response time trend refers to the mean response time of the End Point over the time range. Throughput trend refers to the recent mean throughput time of the End Point. Error rate is the percentage of traces resulting in an HTTP error. Click on markers (points) in the chart to view Trace data that occurred during that time range.

- The **RESPONSE TIME HISTOGRAM** is an interactive graph that shows how many invocations occurred within a given time period. The Y-axis represents the response time of an invocation. The X-axis represents the number of invocations. Using the histogram is an easy way to identify outliers in your data. The longest-running invocations are always at the top of the histogram. If extreme outliers exist, they are indicated by red bars.

- The **Health** of the End Point is determined by the Response Time for requests made over the given time interval. The response times are broken down into various Health Zones (such as *frustrated* , *tolerated* or *satisfied* . Click on a particular Health Zone to see representative Traces within that zone.

6. Click a bar in the histogram or on one of the markers in the Throughput or Response Time Trend graphs. A **REPRESENTATIVE TRACES** panel shows representative traces for some invocations that occurred during the selected duration.

RESPONSE TIME HISTOGRAM




REPRESENTATIVE TRACES **38 ms - 66 ms**

| DURATION | LABEL |
|---|---|
| 43 ms | GET /travel/hotels/search |
| 51 ms | GET /travel/hotels/search |
| 55 ms | GET /travel/hotels/search |
| 40 ms | GET /travel/hotels/search |
| 53 ms | GET /travel/hotels/search |
| 40 ms | GET /travel/hotels/search |

TRACE DETAIL **3:25:34 PM (43 ms)**

- GET /travel/hotels/search 43 m
  - Filter: httpMethodFilter 42 m
    - Filter: springSecurity 42 m
      - Filter: FilterChainProxy 41 m
        - Filter: SecurityContextPersistenceFilter 41 m
          - Filter: LogoutFilter 41 m
            - Filter: UsernamePasswordAuthenticationFilter 41 m
              - Filter: RequestCacheAwareFilter 40 m
                - Filter: SecurityContextHolderAwareRequestFilter 40 m
                  - Filter: AnonymousAuthenticationFilter 39 m
                    - Filter: SessionManagementFilter 39 m
                      - Filter: ExceptionTranslationFilter 38 m
                        - Filter: FilterSecurityInterceptor 37 m
                          - Spring Web Dispatch 36 m
                            - HotelsController#search 16 m
                              - Transaction: BookingService.findBookings (R 14 m
                                - JpaBookingService#findBookings 11 m
                                  - JDBC SELECT (BOOKING) 1 m
                            - Resolve view "hotels/search" 0 m

The **REPRESENTATIVE TRACES** panel includes similar data as that of the **Recent Activity** screen. From here it is easy to drill into the shortest or fastest running traces to see what made them different. See Viewing Recent Activity of Your Application for detailed information about traces and trace details.

## Viewing Recent Activity of Your Application

The following procedure describes how to get an overview of the recent activity of a particular application, or for all applications currently deployed, and then how to drill down to the details of particular application event.

1. From the main Spring Insight dashboard, click the **Recent Activity** tab:


   The **Recent Activity** screen displays traces from your application. A *trace* is a breakdown of the activity of a request. The Recent Activity screen answers the question *What just happened?*

2. In the top-right **Application Selector** drop-down box, select the name of your application.

   Use the **Application Selector** to view activity for *all* applications or for just a single one. Viewing all applications at once is useful when you deploy many Web applications and are interested in activity across the entire set. Viewing a single application filters out activity not related to that application.

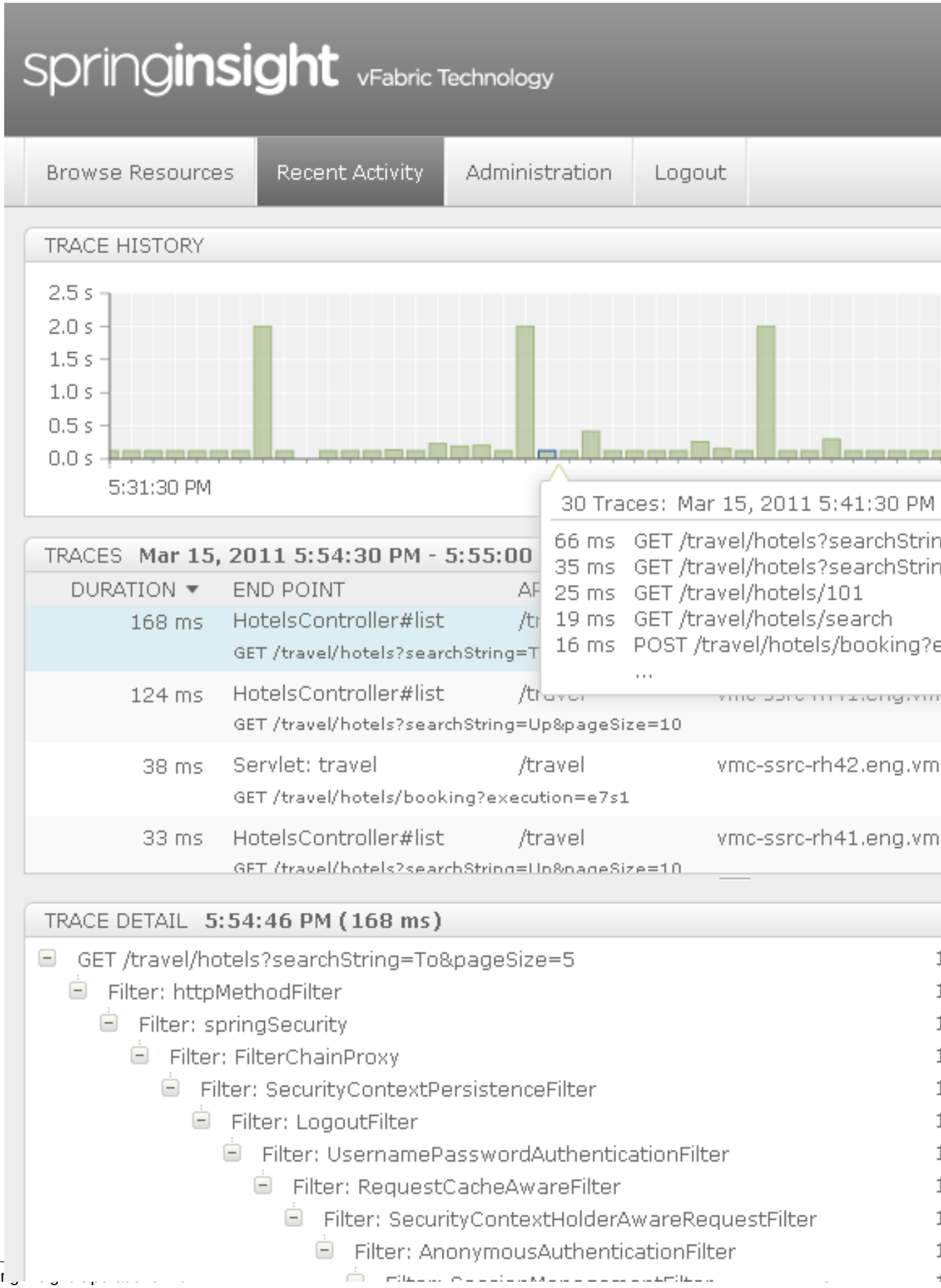
   The **Trace History** panel shows a timeline of the recent activity of your application as real-time requests, represented by bars in the chart. Each bar represents all requests that occurred within a time slice, as measured by the chart. The full time range has 60 time slices. The height of the bar is equal to the longest request that occurred during that window.

   The **Trace History** graph shows activity that Spring Insight has monitored over the past **N** minutes. When a trace is captured by Spring Insight, it shows on the graph as a bar on the right-hand side. As time passes, bars move left until they fall out of the time range. Click on bars to drill deeper.

3. Click on a bar in the **Trace History** chart.

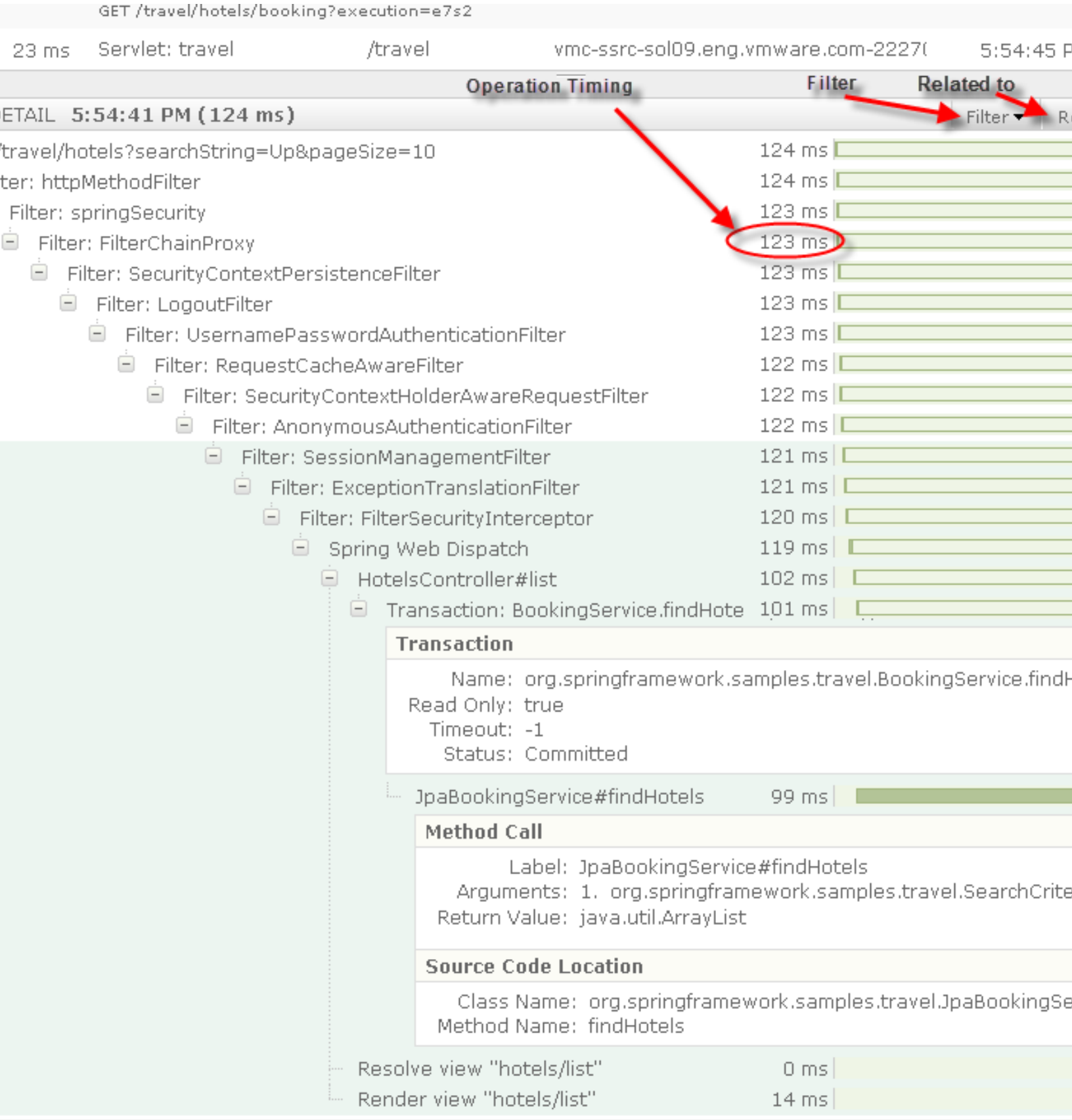
Spring Insight displays a list of traces in the **Traces** panel that executed during the window of time (or time slice), and then details about a specific trace. Insight automatically shows details about the first trace in the **Trace Details** panel. If you click on a different trace, Insight refreshes the details panel with corresponding information.

The Spring Insight dashboard shows the Traces and Trace Detail panels.


springinsight vFabric Technology
Browse Resources
Recent Activity
Administration
Logout
TRACE HISTORY
2.5 s
2.0 s
1.5 s
1.0 s
0.5 s
0.0 s
5:31:30 PM
30 Traces: Mar 15, 2011 5:41:30 PM
66 ms GET /travel/hotels?searchStrin
35 ms GET /travel/hotels?searchStrin
25 ms GET /travel/hotels/101
19 ms GET /travel/hotels/search
16 ms POST /travel/hotels/booking?e
...
TRACES Mar 15, 2011 5:54:30 PM - 5:55:00
DURATION
END POINT
168 ms HotelsController#list
GET /travel/hotels?searchString=T
124 ms HotelsController#list
GET /travel/hotels?searchString=Up&pageSize=10
38 ms Servlet: travel /travel vmc-ssrc-rh42.eng.vm
GET /travel/hotels/booking?execution=e7s1
33 ms HotelsController#list /travel vmc-ssrc-rh41.eng.vm
TRACE DETAIL 5:54:46 PM (168 ms)
GET /travel/hotels?searchString=To&pageSize=5
Filter: httpMethodFilter
Filter: springSecurity
Filter: FilterChainProxy
Filter: SecurityContextPersistenceFilter
Filter: LogoutFilter
Filter: UsernamePasswordAuthenticationFilter
Filter: RequestCacheAwareFilter
Filter: SecurityContextHolderAwareRequestFilter
Filter: AnonymousAuthenticationFilter

You can sort the traces in the **Traces** panel by Duration (how long did the trace take?), Start (when did the trace start?), End Point (what was the request?), or Error (did the request result in an error?)

4. Click on a trace in the **Traces** panel to view its details in the **Trace Detail** panel.


GET /travel/hotels/booking?execution=e7s2
23 ms
Servlet: travel
/travel
vmc-ssrc-sol09.eng.vmware.com-2227(
5:54:45 P
Operation Timing
Filter
Related to
ETAIL 5:54:41 PM (124 ms)
Filter
/travel/hotels?searchString=Up&pageSize=10 124 ms
ter: httpMethodFilter 124 ms
Filter: springSecurity 123 ms
Filter: FilterChainProxy 123 ms
Filter: SecurityContextPersistenceFilter 123 ms
Filter: LogoutFilter 123 ms
Filter: UsernamePasswordAuthenticationFilter 123 ms
Filter: RequestCacheAwareFilter 122 ms
Filter: SecurityContextHolderAwareRequestFilter 122 ms
Filter: AnonymousAuthenticationFilter 122 ms
Filter: SessionManagementFilter 121 ms
Filter: ExceptionTranslationFilter 121 ms
Filter: FilterSecurityInterceptor 120 ms
Spring Web Dispatch 119 ms
HotelsController#list 102 ms
Transaction: BookingService.findHote 101 ms
Transaction
Name: org.springframework.samples.travel.BookingService.findH
Read Only: true
Timeout: -1
Status: Committed
JpaBookingService#findHotels 99 ms
Method Call
Label: JpaBookingService#findHotels
Arguments: 1. org.springframework.samples.travel.SearchCrite
Return Value: java.util.ArrayList
Source Code Location
Class Name: org.springframework.samples.travel.JpaBookingSe
Method Name: findHotels
Resolve view "hotels/list" 0 ms
Render view "hotels/list" 14 ms

The **Trace Detail** panel contains a breakdown of a trace's activity in a tree format. A trace consists of a top-level operation, usually a Web request, and all nested operations.

Spring Insight uses "smart collapse" to determine how to collapse the tree of trace details so that you do not get pages and pages of trace information. You can, of course, uncollapse operations to drill down into their details.

Operations are the fundamental building blocks of traces. An operation can represent a Web request, a transaction, a call to an MVC controller, a file opening, a service request, and so on. Each operation may have other operations nested within it. The nesting structure shows the normal stack-based method invocation pattern.

The operation timing graph shows two pieces of information:

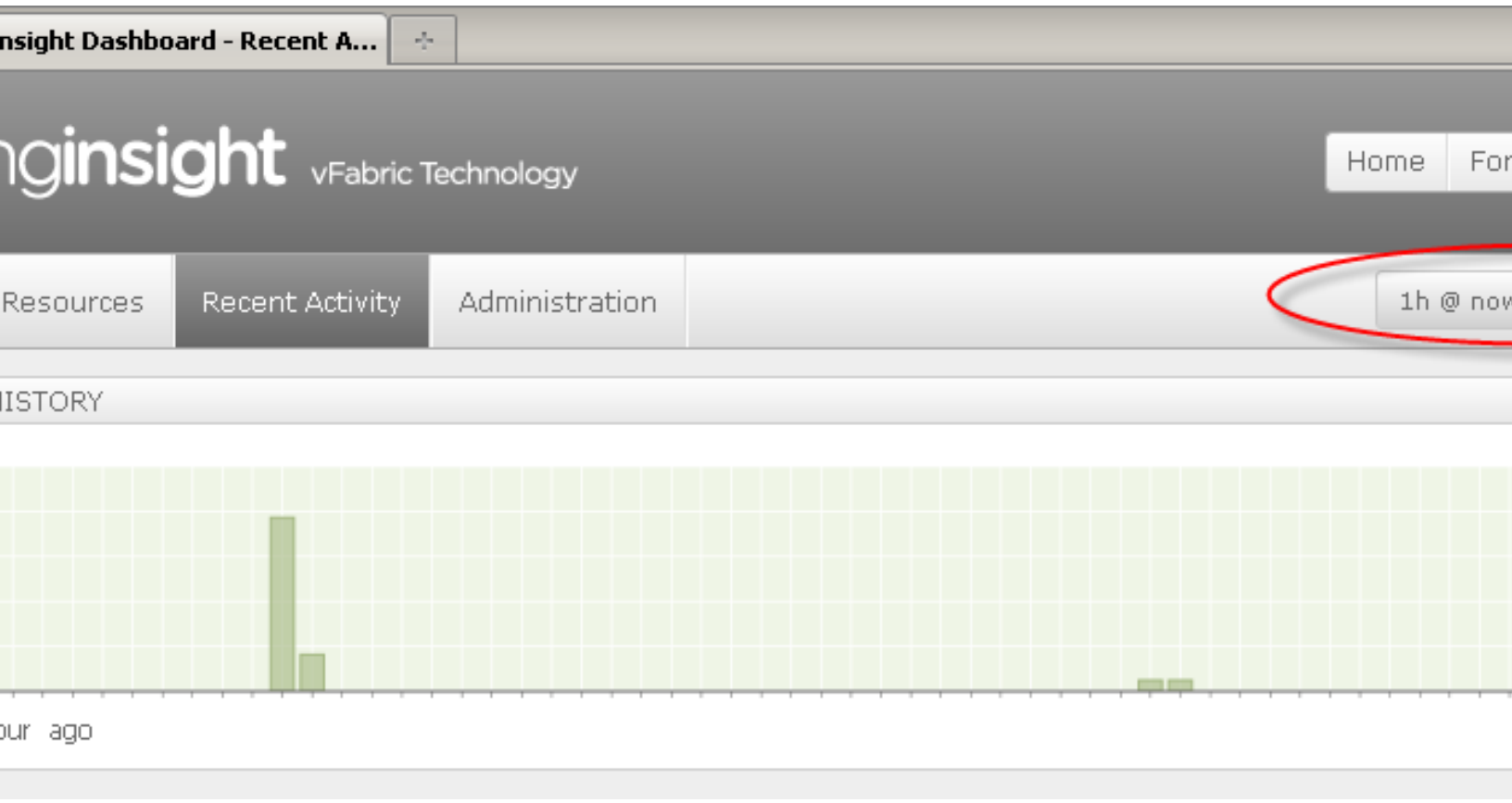
- When the operation executed in relation to the other operations. This information shows whether the operation executed towards the end of the request or the beginning by the location of the green bar in relation to its borders.
- How long the operation took to execute, indicated by the width of the green bar. Because operations can be nested within operations, the green bar shows only how long the particular operation took, *note* the sum of duration of the nested operations. This way you can scan all the nested operations and find the particular one that took the longest time, based on the width of its green bar.

Click on an operation's label, or the entire row, to drill further into the operation details. The details of this panel are specific to each type of operation, and contain the finest granularity of collected data.

Click on **Filter** to filter the trace details shown in the panel. See Filtering Trace Details for additional details about this feature.

Click on **Related To** to navigate to the Trace's corresponding End Point or application in the **Browse Resources** tab. This button is useful when you want to see how similar requests (requests to the same End Point or application) have performed over time.

5. The buttons above the Application Selector help you control the information you see in the Spring Insight dashboard and perform additional administrative tasks:




- Change the global time range using the first drop-down list. The time range specifies how many minutes worth of data shows up in the **Trace History** graph.
- Play or pause the graph movement by specifying `now` or `pause` in the first drop-down list. If the graph is in Play mode, the word `Live` appears under the right hand side of the graph. If you have paused the graph movement, the time when you clicked the pause button appears instead.
- Click the << or >> buttons to rewind or fast-forward, based on the specified time ranges. Spring Insight persists all trace information about all your applications to disk, which means you can rewind and look at trace information from when you first began to track the performance of your application.
- In the right-most drop-down list, choose `Refresh` to refresh the trace history by reloading all data within the **Trace History** graph.

## Filtering Trace Details

Depending on the nature of a specific trace, the list of operations in the corresponding **Trace Detail** pane might be very long. Spring Insight uses smart collapse, which means it collapses those operations it thinks are unimportant but uncollapses operations which are most likely interesting to you. However, the **Trace Details** pane could still be very large. In this case, you might want to filter the list of operations so as to display only certain types that you are interested in.

The list of available filters depends on the plug-ins currently installed in Spring Insight as well as the type of operations in the current list of trace details. Spring Insight has a number of plug-ins installed by default. Click the **Administration** tab on the main dashboard then **Collection Plug-Ins** to see the list. If you have previously added your own custom plug-in to Spring Insight so as to display custom trace details, then this filter might also be available.

The default list of filters is as follows:

- **Database**: Filters operations based on whether they are related to general database calls. This could include transactions as well as standard calls to a relational database, for example.
- **JDBC**: Filters operations based on whether they are JDBC calls. The results of this filter are a subset of the results of the Databse filter.
- **Web**: Filters operations based on whether they are related to Web calls, such as HTTP requests and responses, rendering of HTML pages, and so on.

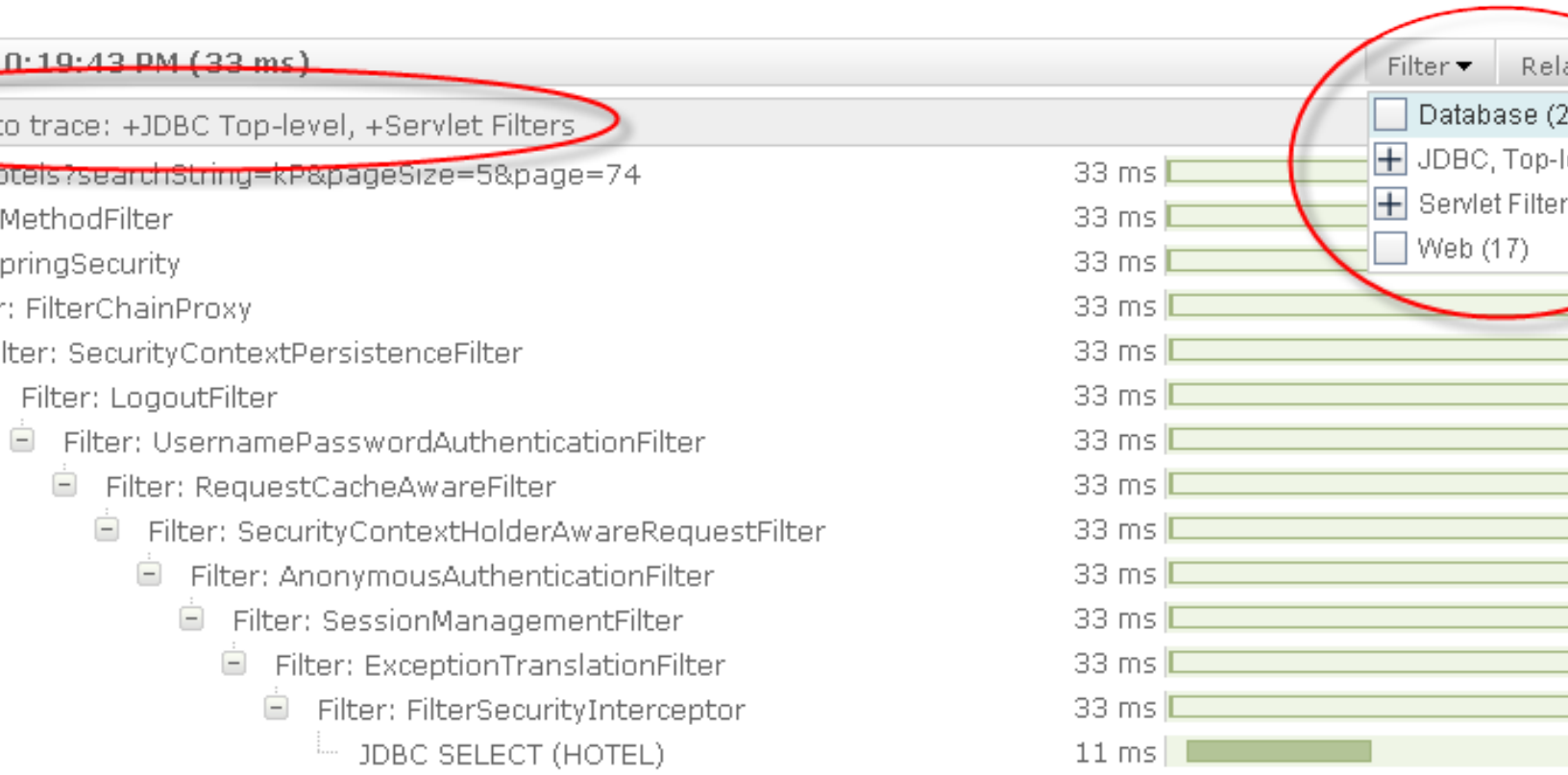
Filters are tri-state—you can set each filter to one of three different states:

- Unselected – operations are included in the results unless one or more other filters are set to + (plus)
- + (plus sign) – only these operations are included in the results
- - (minus sign) - only these operations are excluded from the results

Filters are *sticky*, which means that they stay in place even when you navigate away from the page in which you set the filter. This is useful if you want to look at multiple applications or End Points, searching for JDBC calls, because you do not have to reset the filter each time you look at a different trace. You can remove the filter at any time, as described in the procedure below.

To use filters, follow these steps:

1. Display the **Trace Details** pane for a particular trace you are interested in. See Viewing Recent Activity of Your Application.
2. In the top-right corner of the **Trace Details** pane, click the **Filter** drop-down list and select the filters you want to apply. You can select multiple filters; the results are a sum of the individual filter results. Set the filter to + (plus) to include only those operations, - (minus) to exclude the operations, or leave the box unselected. If you do not set a filter to + or -, the operations are included unless another filter is set to +.



The preceding graphic shows how to apply a JDBC filter.

3. Click to the left of the **Filter** drop-down list to actually apply the filter. A message appears below the **Trace Detail** label that alerts that a filter is being applied.

   After applying the filter, the trace details navigation tree should have gotten smaller, and the only details that are now included are those related to the specified filter, such as JDBC calls in our example.

   Because of the stickiness of the filter, as you navigate away from your current page you'll see that Spring Insight continues to apply the filter.

4. To remove the filter, click on the `X` to the left of the `Filter applied to trace` alert.

## Exporting and Importing Traces

With this feature, you can export a trace to a file and then import it into either the same or different instance of Spring Insight. This feature is useful if the Insight user who captured the trace in real time wants to look at it at a later time, or hand it off to another person.

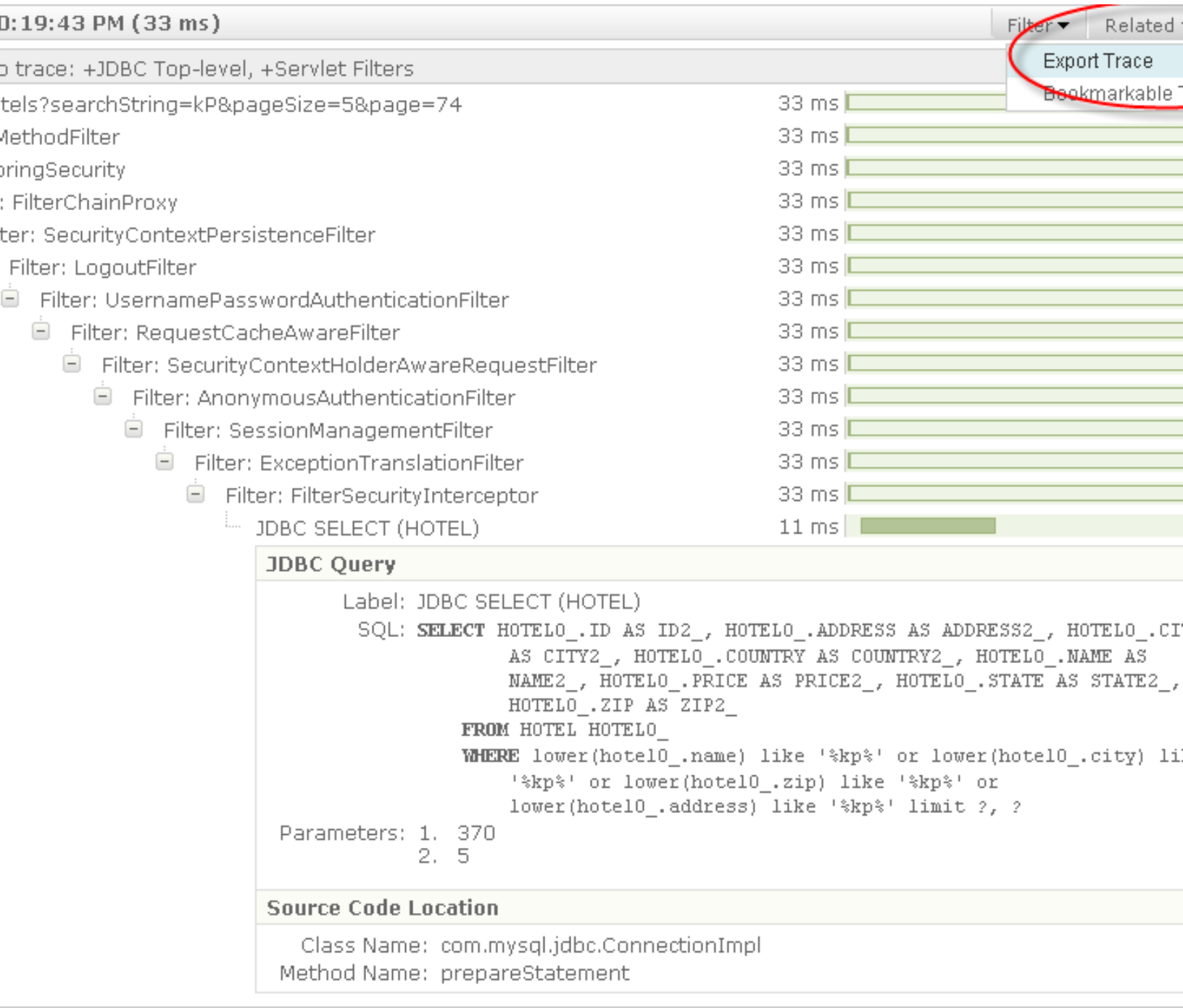
For example, assume that a QA engineer is testing their company's Web application and runs into an error, or an operation takes an excessive time to complete. The QA engineer uses Spring Insight to take a look at what just happened by navigating to the relevant trace and viewing its details. If the QA engineer decides to open a bug about the problem, they can export the trace to a file and then attach this file to the bug issue so that the developer can take a look at a later date. The developer does not need to actually deploy the application; rather, they can simply look at the trace details to figure out which operation caused the error, where the excessive time occurred, and so on. This is an easy way to capture a complete set of trace information about an error event, even if the event is not reliably reproduceable.

You can only import traces that were previously exported from a Spring Insight instance.

**To export a trace:**

1. Display the **Trace Details** pane for the particular trace you want to export. See Viewing Recent Activity of Your Application.

2. From the right-most drop-down list in the **Trace Details** pane  , select **Export Trace**:




3. Save the exported trace to a file. The name of the file will be of the form `trace-`*`identifier`*`.insight`.

**To import a trace:**

1. From the main Spring Insight dashboard, click the **Administration** tab.
2. In the left pane, click **Import Trace**.
3. In the right pane, use the browse button to browse the trace you want to import; the name of the file will be of the form `trace-`*`identifier`*`.insight`.

4. Click **Import**.

5. Spring Insight immediately takes you to the **Recent Activity** page with the trace and its details displayed. To locate the trace in the **Trace History** pane, go back in time to when the trace was originally exported.

## Customizing End Point Thresholds

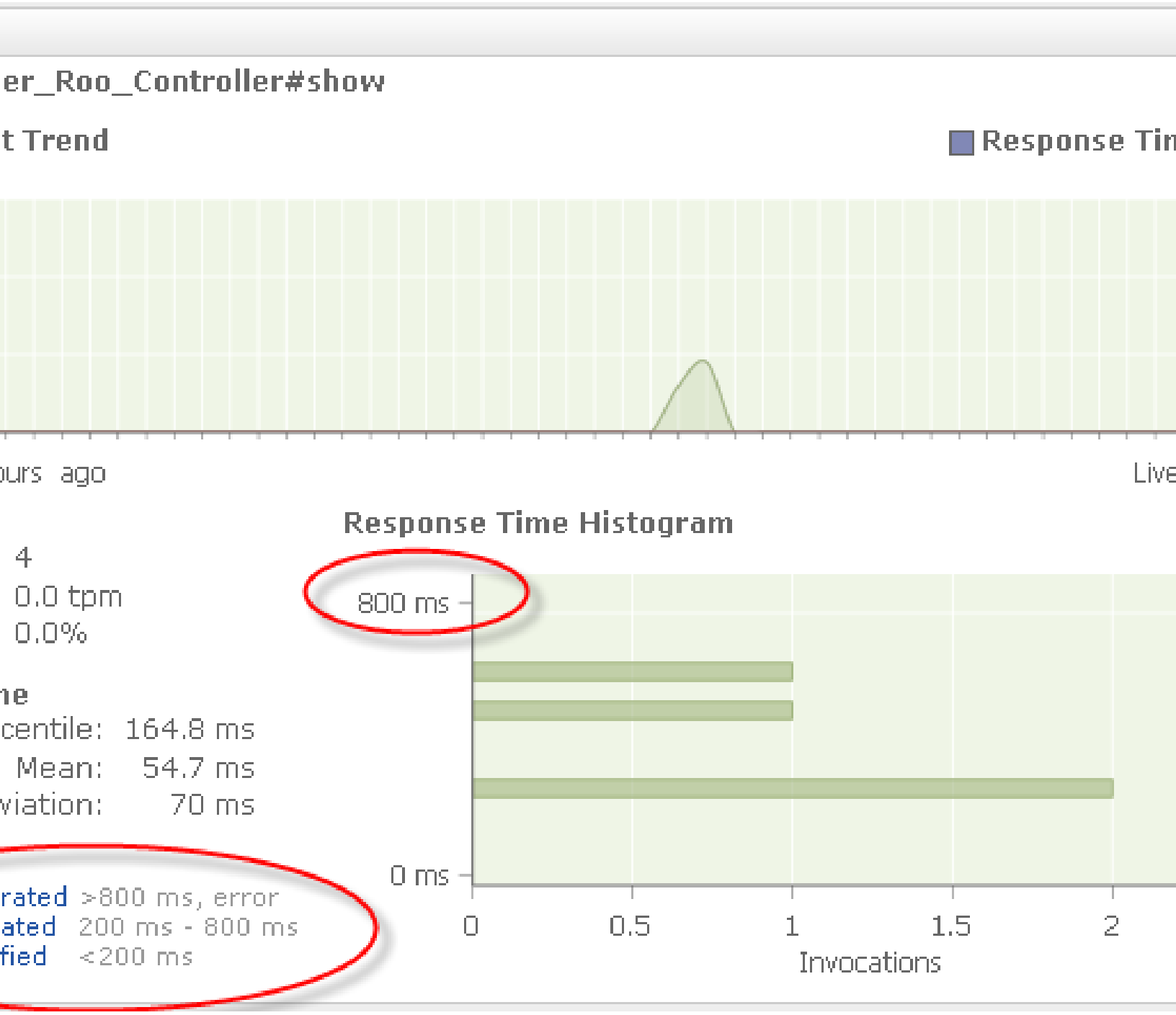
The health of an end point is based on how many traces took longer to execute than the response time threshold.

By default, Spring Insight uses a response time threshold of 200ms. In the response time histogram, the upper limit of the y-axis is 4-times the threshold, or 800 ms by default. The time chunks of the y-axis are not evenly distributed, but rather, broken up in a way to show the distribution of the response times of the recent End Point traces.

If, for a given trace or subset of traces, you find that the default threshold is too high or too low, you can change it. For example, if you find that all your response times for the `.*show.*` methods are almost always below 100 ms, you might want to set this as the threshold. The histogram will then have a smaller range, and thus show more fine grained information. Similarly, if you have an end point in which the response times are always over 200 ms, the health of this end point will almost always show as `frustrated`. If you decide that it is acceptable if a threshold of 300ms is acceptable, then you can change it for this End Point so it will show as appropriately healthy.

In other words, if you change the response time threshold for an End Point, you change the criteria that Spring Insight uses to decide whether it is healthy or not.

The following graphic shows the health and response time histogram for an End Point whose threshold is the default (200ms):




Note the upper limit in the histogram of 800ms, and the `satisfied` range is under 200ms. A response time of over 200ms but under 800ms is tolerated, but over 800ms is frustrated, or unhealthy.

Note that all response times are under the default threshold so the End Point is healthy. If the response times are all significantly below the default threshold, it might be a good candidate to lower the threshold to get more fine-grained response time information.

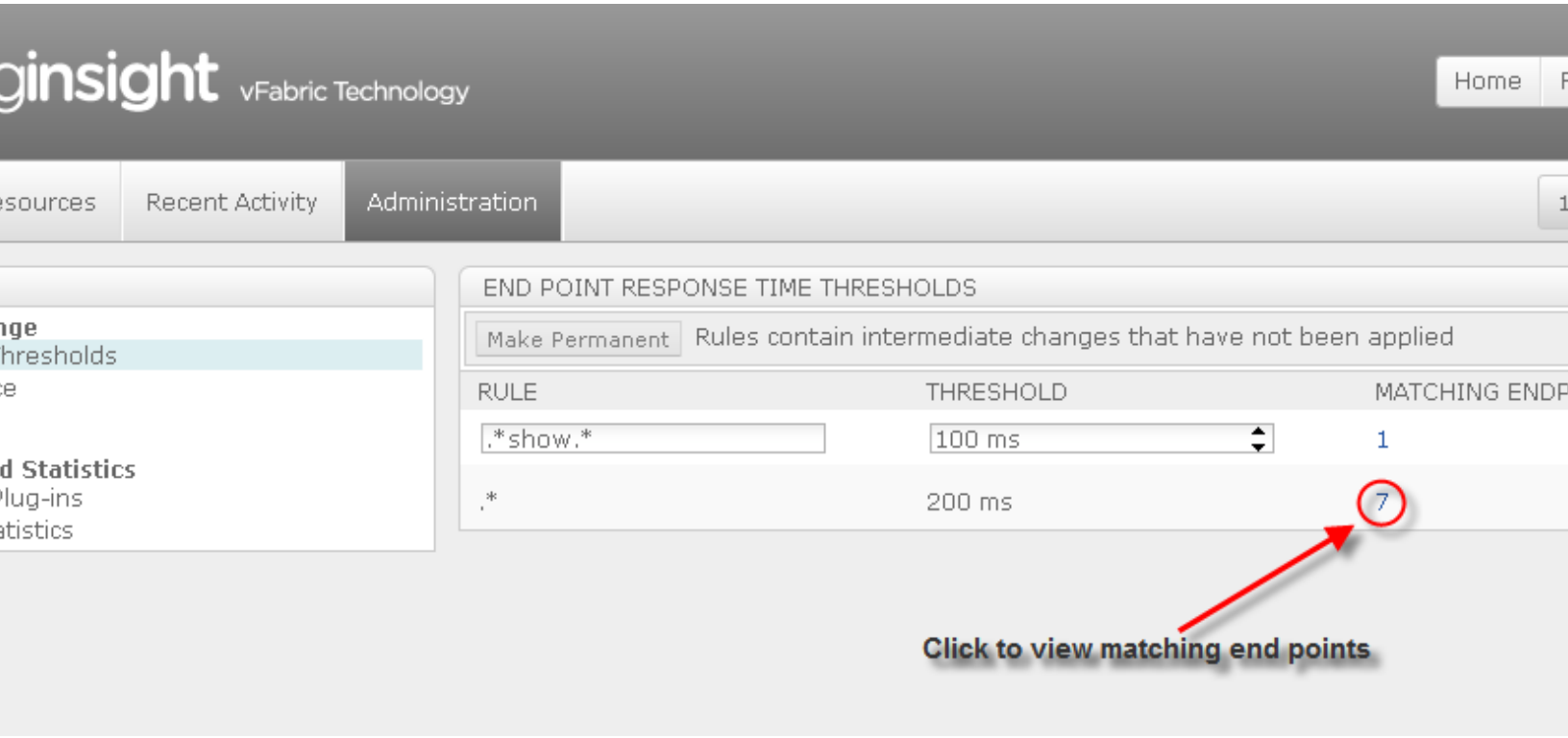
To change the threshold for an End Point or set of End Points:

1. From the main Spring Insight dashboard, click the **Administration** tab.

2. Click **End Point Threshold** in the left pane.

3. In the right pane, click **New**:

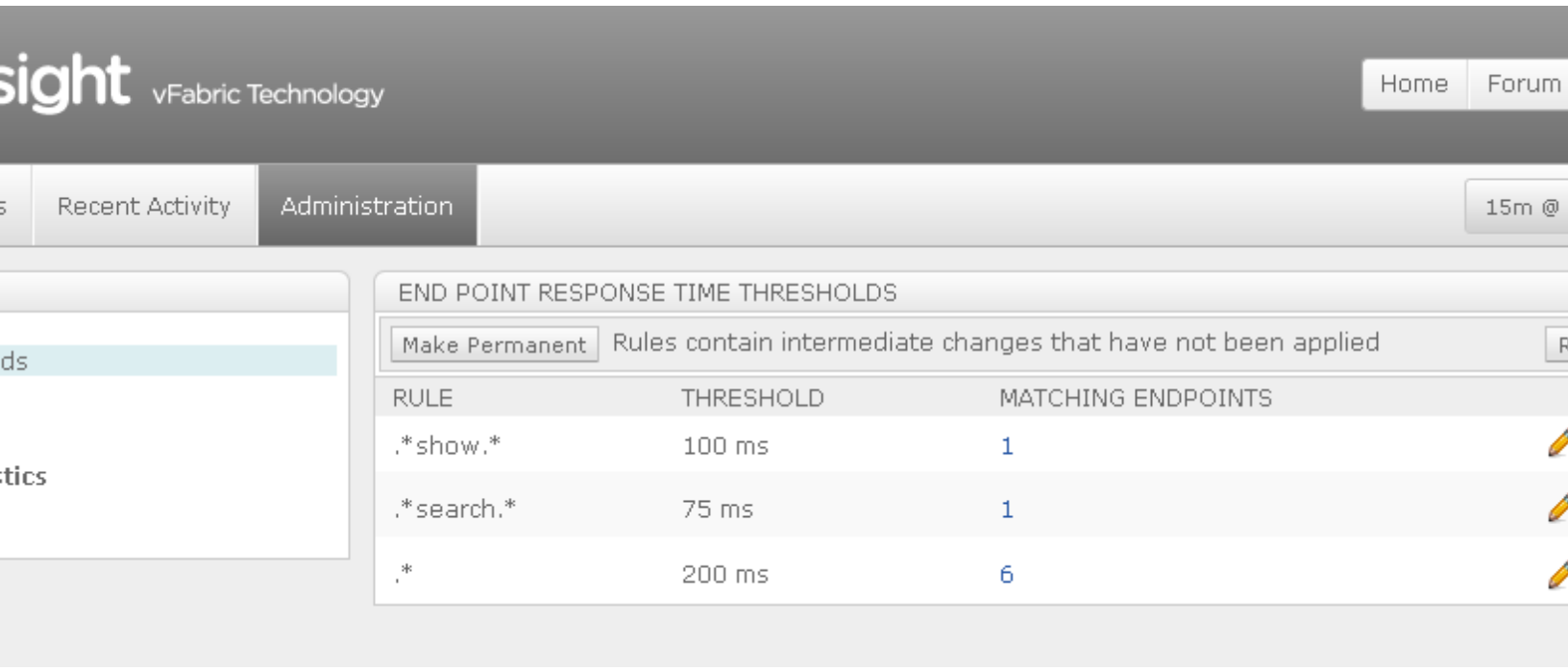
4. In the **Rule** field, enter a regular expression that corresponds to the End Point or End Points for which you want to change the threshold. For example, if you want to specify all `show` methods, enter `.*show.*`:




5. Enter the new threshold. As described above, the default Spring Insight threshold is `200` milliseconds.

6. Click **Save**. The **Matching End Points** column automatically shows the number of End Points that match this rule; the number is a link. Click on this link to see the list of matching End Points.

7. Click the **Make Permanent** to apply the change. The new rule appears in the table:



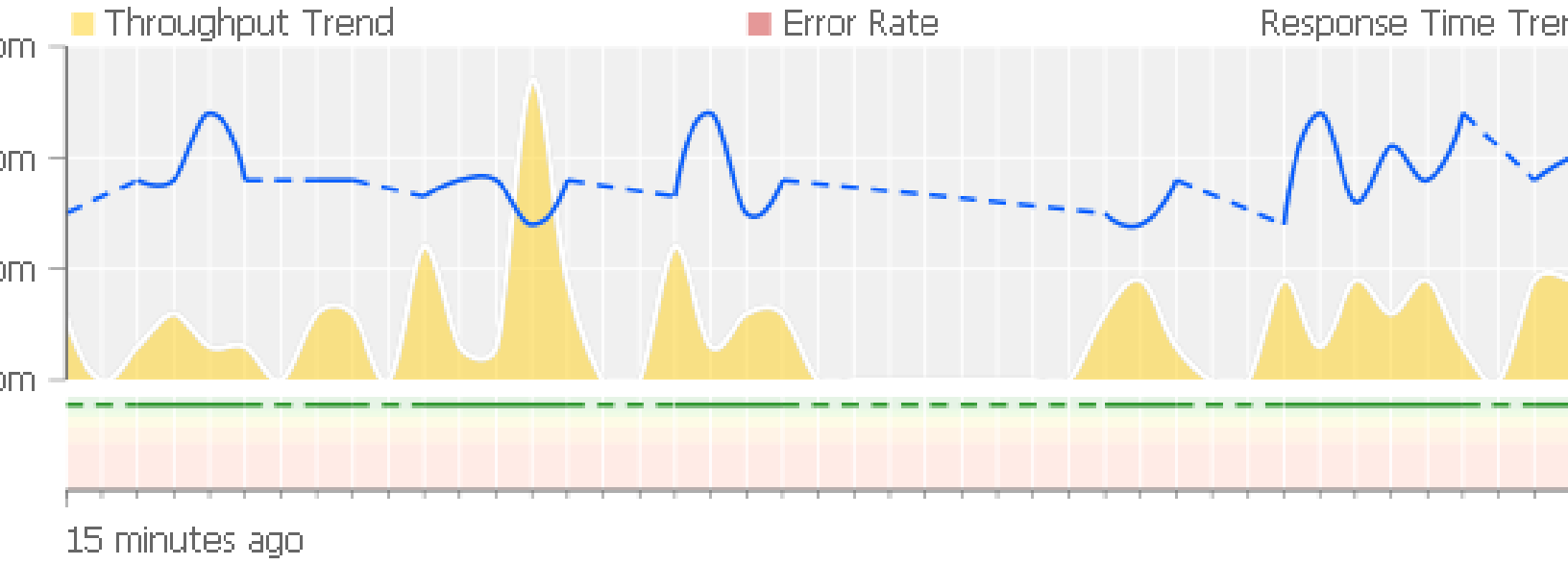

8. If you have more than three rules, you can use the up and down arrow buttons to change the order in which Spring Insight applies the rules. Insight applies the rules from first in the list to last. The default rule (`.*`) should always be last.

9. Browse to a trace that matches the rule. Note that the y-axis of the Response Time Histogram now has an upper limit of 4-times the new threshold. The health of the End Point is now satisfied when its response time is below the new threshold. In our example, the new threshold is 100 ms and so the upper limit is 400 ms:

INT

ontroller#show

el/hotels/112

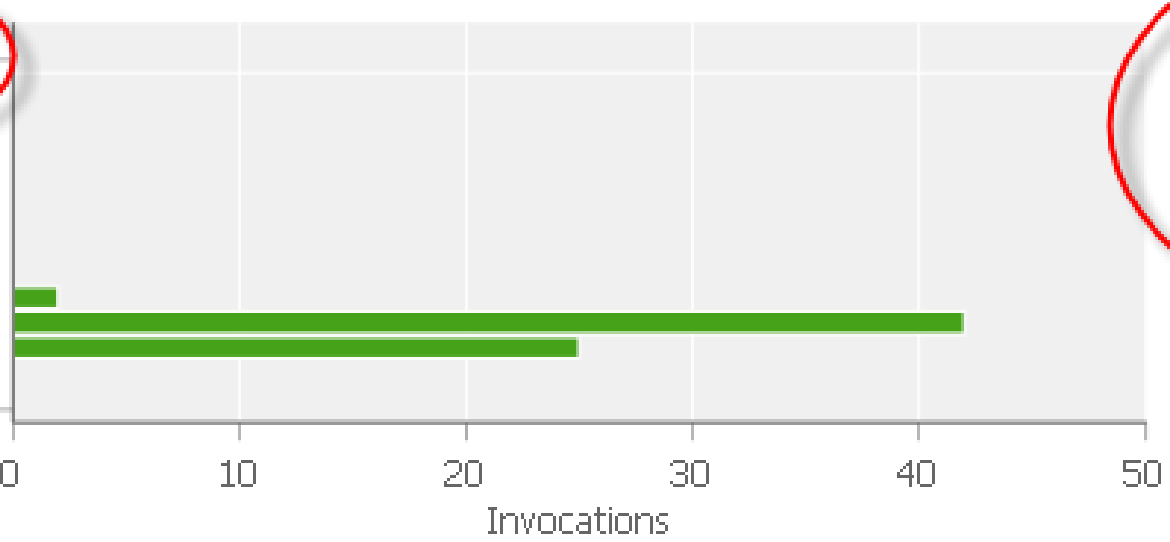



| | |
|---|---|
| **Throughput** | 4.6 tpm |
| **Invocations** | 69 |
| **Errors** | 0 (0.0%) |

RESPONSE TIME

| | |
|---|---|
| **95th Percentile** | 4 ms |
| **Mean** | 2 ms |
| **Standard Deviation** | 0 ms |

NSE TIME HISTOGRAM




**Health Legend:**

**0** Frustrated >400

**0** Tolerated 100 n

**69** Satisfied <100

## Viewing Spring Insight Data In Google Speed Tracer

Speed Tracer is a Google Chrome extension that analyzes how your application is performing inside the browser. It measures how long the browser takes to render, transform CSS, show images, process events, and so on.

Although Speed Tracer is a great tool for determining where CPU time is spent within the browser process, it cannot see into what the application itself is doing in the back end. For that, it needs Insight. The two products are now integrated so that you can now see Trace data interleaved with Speed Tracer's browser timings.

To see Insight Trace data within Speed Tracer for an application deployed to Spring Insight, follow these steps:

1. Deploy your application to a tc Runtime instance that is configured with Spring Insight.

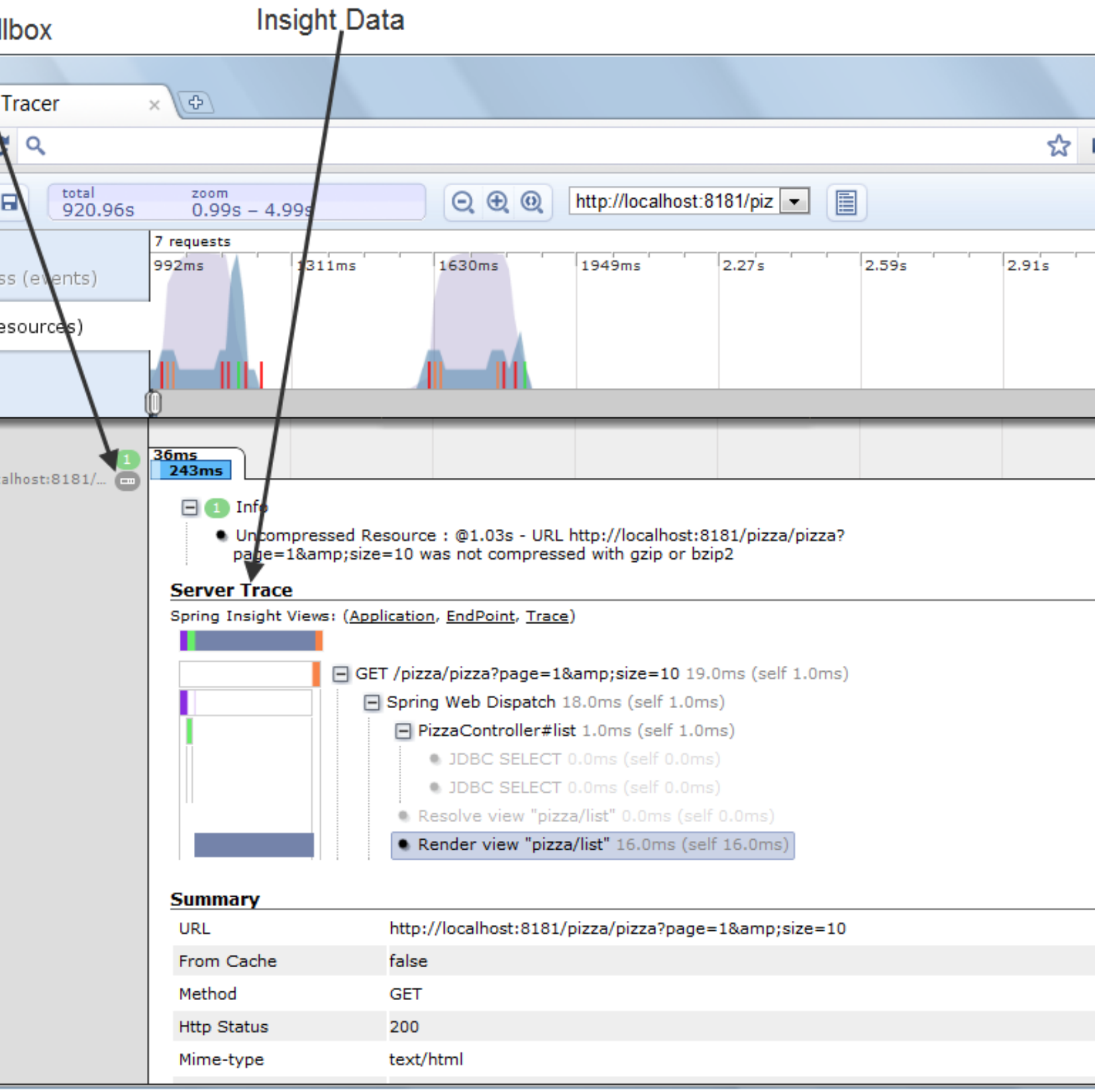
2. If you have not already done so, download the version of Google Chrome that has been instrumented for Speed Tracer, install the Speed Tracer extension, and launch the Chrome browser with the appropriate flag. For details, see Getting Started with Speed Tracer.

3. Open up the Speed Tracer console by clicking on the stopwatch icon in the top-right corner of the Google Chrome browser.

4. Using Google Chrome, navigate to a page of your deployed application and perform some action.

5. In Speed Tracer, click on the **Network (resources)** timeline. In the left column, search for resources that have a grey pillbox, with tooltip `Includes timing data from the server`; these resources include Insight data along with the standard Speed Tracer data. See the graphic in the next step.

6. Expand the resource to view the Insight data, listed under the **Server Trace** section. This section includes a brief summary of the Trace frame stack and allows easy navigation into various parts of Spring Insight related to the given Trace. To see more detailed information, select the **Trace**, **End Point** or **Application** links, which will jump into Insight at the appropriate location so you can further drill down.


Insight Data
Tracer
total
920.96s
zoom
0.99s – 4.99s
http://localhost:8181/piz
7 requests
992ms
1311ms
1630ms
1949ms
2.27s
2.59s
2.91s
36ms
243ms
1 Info
Uncompressed Resource : @1.03s - URL http://localhost:8181/pizza/pizza?page=1&size=10 was not compressed with gzip or bzip2
Server Trace
Spring Insight Views: (Application, EndPoint, Trace)
GET /pizza/pizza?page=1&size=10 19.0ms (self 1.0ms)
Spring Web Dispatch 18.0ms (self 1.0ms)
PizzaController#list 1.0ms (self 1.0ms)
JDBC SELECT 0.0ms (self 0.0ms)
JDBC SELECT 0.0ms (self 0.0ms)
Resolve view "pizza/list" 0.0ms (self 0.0ms)
Render view "pizza/list" 16.0ms (self 16.0ms)
Summary
URL
http://localhost:8181/pizza/pizza?page=1&size=10
From Cache
false
Method
GET
Http Status
200
Mime-type
text/html