

# Spring Cloud GCP Reference Documentation

1.0.0.RC1

João André Martins , Jisha Abubaker , Ray Tsang , Mike Eltsufin ,  
Artem Bilan , Andreas Berger , Balint Pato , Chengyuan Zhao

Copyright © 2017-2018Pivotal Software, Inc.

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

# Table of Contents

1. Introduction .....	1
2. Dependency Management .....	2
3. Getting started .....	3
3.1. Spring Initializr .....	3
3.2. Code Samples .....	3
3.3. Code Challenges .....	3
3.4. Getting Started Guides .....	3
4. Spring Cloud GCP Core .....	4
4.1. Project ID .....	4
4.2. Credentials .....	4
Scopes .....	5
Spring Initializr .....	6
5. Spring Cloud GCP for Pub/Sub .....	7
5.1. Pub/Sub operations abstraction .....	7
Publishing to a topic .....	7
Subscribing to a subscription .....	8
Pulling messages from a subscription .....	8
5.2. Pub/Sub management .....	8
Creating a topic .....	9
Deleting a topic .....	9
Listing topics .....	9
Creating a subscription .....	9
Deleting a subscription .....	10
Listing subscriptions .....	10
5.3. Configuration .....	10
6. Spring Resources .....	14
6.1. Google Cloud Storage .....	14
6.2. Configuration .....	15
7. Spring JDBC .....	16
7.1. Prerequisites .....	16
7.2. Spring Boot Starter for Google Cloud SQL .....	16
DataSource creation flow .....	17
Troubleshooting tips .....	18
Connection issues .....	18
Errors like <code>c.g.cloud.sql.core.SslSocketFactory : Re-throwing</code> cached exception due to attempt to refresh instance information too soon after error .....	18
PostgreSQL: <code>java.net.SocketException: already connected</code> issue .....	18
8. Spring Integration .....	19
8.1. Channel Adapters for Google Cloud Pub/Sub .....	19
Inbound channel adapter .....	19
Outbound channel adapter .....	20
Header mapping .....	21
8.2. Channel Adapters for Google Cloud Storage .....	21
Inbound channel adapter .....	22
Inbound streaming channel adapter .....	22
Outbound channel adapter .....	23

9. Spring Cloud Stream .....	24
9.1. Overview .....	24
9.2. Configuration .....	24
10. Spring Cloud Sleuth .....	25
10.1. Tracing .....	25
10.2. Spring Boot Starter for Stackdriver Trace .....	26
10.3. Integration with Logging .....	27
11. Stackdriver Logging Support .....	28
11.1. Web MVC Interceptor .....	28
11.2. Logback Support .....	29
Log via API .....	29
Log via Console .....	29
12. Cloud Foundry .....	32

# 1. Introduction

The Spring Cloud GCP project aims at making the Spring Framework a first-class citizen of Google Cloud Platform (GCP).

Currently, Spring Cloud GCP lets you leverage the power and simplicity of the Spring framework to:

1. Publish and subscribe from Google Cloud Pub/Sub topics
2. Configure Spring JDBC with a few properties to use Google Cloud SQL
3. Write and read from Spring Resources backed up by Google Cloud Storage
4. Exchange messages with Spring Integration using Google Cloud Pub/Sub on the background
5. Trace the execution of your app with Spring Cloud Sleuth and Google Stackdriver Trace
6. Consume and produce Google Cloud Storage data via Spring Integration GCS Channel Adapters

## 2. Dependency Management

The Spring Cloud GCP Bill of Materials (BOM) contains the versions of all the dependencies it uses.

If you're a Maven user, adding the following to your pom.xml file will allow you to not specify any Spring Cloud GCP dependency versions. Instead, the version of the BOM you're using determines the versions of the used dependencies.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-gcp-dependencies</artifactId>
      <version>1.0.0.RC1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

In the following sections, it will be assumed you are using the Spring Cloud GCP BOM and the dependency snippets will not contain versions.

Gradle users can achieve the same kind of BOM experience using Spring's [dependency-management-plugin](#) Gradle plugin. For simplicity, the Gradle dependency snippets in the remainder of this document will also omit their versions.

## 3. Getting started

There are many available resources to get you up to speed with our libraries as quickly as possible.

### 3.1 Spring Initializr

There are three entries in [Spring Initializr](#) for Spring Cloud GCP:

- GCP Support
- GCP Messaging
- GCP Storage

The GCP Support entry contains auto-configuration support for every Spring Cloud GCP integration. Most of the autoconfiguration code is only enabled if other dependencies are added to the classpath.

Spring Cloud GCP Starter	Required dependencies
Logging	org.springframework.cloud:spring-cloud-gcp-starter-logging
SQL - MySql	org.springframework.cloud:spring-cloud-gcp-starter-sql-mysql
SQL - PostgreSQL	org.springframework.cloud:spring-cloud-gcp-starter-sql-postgres
Trace	org.springframework.cloud:spring-cloud-gcp-starter-trace

The GCP Messaging entry adds the GCP Support entry and all the required dependencies so that the Google Cloud Pub/Sub integrations work out of the box.

The GCP Storage entry adds the GCP Support entry and all the required dependencies so that the Google Cloud Storage integrations work out of the box.

### 3.2 Code Samples

There are [code samples](#) available that demonstrate the usage of all our integrations. [The Vision API sample](#) shows how to use `spring-cloud-gcp-starter` for authentication.

### 3.3 Code Challenges

In a code challenge, you perform a task step by step, using one integration. There are a number of challenges available in the [Google Developers Codelabs](#) page.

### 3.4 Getting Started Guides

A Spring Getting Started guide on messaging with Spring Integration Channel Adapters for Google Cloud Pub/Sub is available from [Spring Guides](#).

## 4. Spring Cloud GCP Core

At the center of every Spring Cloud GCP module are the concepts of `GcpProjectIdProvider` and `CredentialsProvider`.

Spring Cloud GCP provides a Spring Boot starter to auto-configure the core components.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter'
}
```

### 4.1 Project ID

`GcpProjectIdProvider` is a functional interface that returns a GCP project ID string.

```
public interface GcpProjectIdProvider {
    String getProjectId();
}
```

The Spring Cloud GCP starter auto-configures a `GcpProjectIdProvider`. If a `spring.cloud.gcp.project-id` property is specified, the provided `GcpProjectIdProvider` returns that property value.

```
spring.cloud.gcp.project-id=my-gcp-project-id
```

Otherwise, the project ID is discovered based on a [set of rules](#):

1. The project ID specified by the `GOOGLE_CLOUD_PROJECT` environment variable
2. The Google App Engine project ID
3. The project ID specified in the JSON credentials file pointed by the `GOOGLE_APPLICATION_CREDENTIALS` environment variable
4. The Google Cloud SDK project ID
5. The Google Compute Engine project ID, from the Google Compute Engine Metadata Server

### 4.2 Credentials

`CredentialsProvider` is a functional interface that returns the credentials to authenticate and authorize calls to Google Cloud Client Libraries.

```
public interface CredentialsProvider {
    Credentials getCredentials() throws IOException;
}
```

The Spring Cloud GCP starter auto-configures a `CredentialsProvider`. It uses the `spring.cloud.gcp.credentials.location` property to locate the OAuth2 private key of a

Google service account. Keep in mind this property is a Spring Resource, so the credentials file can be obtained from a number of [different locations](#) such as the file system, classpath, URL, etc. The next example specifies the credentials location property in the file system.

```
spring.cloud.gcp.credentials.location=file:/usr/local/key.json
```

If that property isn't specified, the starter tries to discover credentials from a [number of places](#):

1. Credentials file pointed to by the `GOOGLE_APPLICATION_CREDENTIALS` environment variable
2. Credentials provided by the Google Cloud SDK `gcloud auth application-default login` command
3. Google App Engine built-in credentials
4. Google Cloud Shell built-in credentials
5. Google Compute Engine built-in credentials

If your app is running on Google App Engine or Google Compute Engine, in most cases, you should omit the `spring.cloud.gcp.credentials.location` property and, instead, let the Spring Cloud GCP Starter get the correct credentials for those environments. On App Engine Standard, the [App Identity service account credentials](#) are used, on App Engine Flexible, the [Flexible service account credential](#) are used and on Google Compute Engine, the [Compute Engine Default Service Account](#) is used.

## Scopes

By default, the credentials provided by the Spring Cloud GCP Starter contain scopes for every service supported by Spring Cloud GCP.

Service	Scope
Pub/Sub	<a href="https://www.googleapis.com/auth/pubsub">https://www.googleapis.com/auth/pubsub</a>
Storage (Read Only)	<a href="https://www.googleapis.com/auth/devstorage.read_only">https://www.googleapis.com/auth/devstorage.read_only</a>
Storage (Write/Write)	<a href="https://www.googleapis.com/auth/devstorage.read_write">https://www.googleapis.com/auth/devstorage.read_write</a>
Runtime Config	<a href="https://www.googleapis.com/auth/cloudruntimeconfig">https://www.googleapis.com/auth/cloudruntimeconfig</a>
Trace (Append)	<a href="https://www.googleapis.com/auth/trace.append">https://www.googleapis.com/auth/trace.append</a>
Cloud Platform	<a href="https://www.googleapis.com/auth/cloud-platform">https://www.googleapis.com/auth/cloud-platform</a>

The Spring Cloud GCP starter allows you to configure a custom scope list for the provided credentials. To do that, specify a comma-delimited list of [Google OAuth2 scopes](#) in the `spring.cloud.gcp.credentials.scopes` property.

`spring.cloud.gcp.credentials.scopes` is a comma-delimited list of [Google OAuth2 scopes](#) for Google Cloud Platform services that the credentials returned by the provided `CredentialsProvider` support.

```
spring.cloud.gcp.credentials.scopes=https://www.googleapis.com/auth/pubsub,https://www.googleapis.com/auth/sqlservice.admin
```



You can also use `DEFAULT_SCOPES` placeholder as a scope to represent the starters default scopes, and append the additional scopes you need to add.

```
spring.cloud.gcp.credentials.scopes=DEFAULT_SCOPES,https://www.googleapis.com/auth/cloud-vision
```

## Spring Initializr

This starter is available from [Spring Initializr](#) through the `GCP Support` entry.

## 5. Spring Cloud GCP for Pub/Sub

Spring Cloud GCP provides an abstraction layer to publish to and subscribe from Google Cloud Pub/Sub topics and to create, list or delete Google Cloud Pub/Sub topics and subscriptions.

A Spring Boot starter is provided to auto-configure the various required Pub/Sub components.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-pubsub</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-pubsub'
}
```

This starter is also available from [Spring Initializr](#) through the `GCP Messaging` entry.

A [sample application](#) is available.

### 5.1 Pub/Sub operations abstraction

`PubSubOperations` is an abstraction that allows Spring users to use Google Cloud Pub/Sub without depending on any Google Cloud Pub/Sub API semantics. It provides the common set of operations needed to interact with Google Cloud Pub/Sub. `PubSubTemplate` is the default implementation of `PubSubOperations` and it uses the [Google Cloud Java Client for Pub/Sub](#) to interact with Google Cloud Pub/Sub.

`PubSubTemplate` depends on a `PublisherFactory` and a `SubscriberFactory`. The `PublisherFactory` provides a Google Cloud Java Client for Pub/Sub Publisher. The `SubscriberFactory` provides the `Subscriber` for asynchronous message pulling, as well as a `SubscriberStub` for synchronous pulling and an `Acknowledger`, for the cases where messages are automatically acknowledged. The Spring Boot starter for GCP Pub/Sub auto-configures a `PublisherFactory` and `SubscriberFactory` with default settings and uses the `GcpProjectIdProvider` and `CredentialsProvider` auto-configured by the Spring Boot GCP starter.

The `PublisherFactory` implementation provided by Spring Cloud GCP Pub/Sub, `DefaultPublisherFactory`, caches `Publisher` instances by topic name, in order to optimize resource utilization.

#### Publishing to a topic

`PubSubTemplate` provides asynchronous methods to publish messages to a Google Cloud Pub/Sub topic. The `publish()` method takes in a topic name to post the message to, a payload of a generic type and, optionally, a map with the message headers.

Here is an example of how to publish a message to a Google Cloud Pub/Sub topic:

```
public void publishMessage() {
    this.pubSubTemplate.publish("topic", "your message payload", ImmutableMap.of("key1", "val1"));
}
```

By default, the `SimplePubSubMessageConverter` is used to convert payloads of type `byte[]`, `ByteString`, `ByteBuffer`, and `String` to Pub/Sub messages.

For serialization and deserialization of POJOs using Jackson JSON, configure the `PubSubTemplate` to use the `JacksonPubSubMessageConverter` by calling the `setMessageConverter()` method. Alternatively, if you're using the starter and have an instance of the Jackson `ObjectMapper` in the application context, the `JacksonPubSubMessageConverter` will be automatically configured for you.

## Subscribing to a subscription

Google Cloud Pub/Sub allows many subscriptions to be associated to the same topic. `PubSubTemplate` allows you to subscribe to subscriptions via the `subscribe()` method. It relies on a `SubscriberFactory` object, whose only task is to generate Google Cloud Pub/Sub `Subscriber` objects. When subscribing to a subscription, messages will be pulled from Google Cloud Pub/Sub asynchronously, on a certain interval.

The Spring Boot starter for Google Cloud Pub/Sub auto-configures a `SubscriberFactory`.

## Pulling messages from a subscription

Google Cloud Pub/Sub supports synchronous pulling of messages from a subscription. This is different from subscribing to a subscription, in the sense that subscribing is an asynchronous task which polls the subscription on a set interval.

The `pullNext()` method allows for a single message to be pulled and automatically acknowledged from a subscription. The `pull()` method pulls a number of messages from a subscription, allowing for the retry settings to be configured. Any messages received by `pull()` are not automatically acknowledged. Instead, since they are of the kind `AcknowledgeablePubsubMessage`, you can acknowledge them by calling the `ack()` method, or negatively acknowledge them by calling the `nack()` method. The `pullAndAck()` method does the same as the `pull()` method and, additionally, acknowledges all received messages.

In order to acknowledge or negatively acknowledge the messages received from `pull()`, you can use the `acknowledger` provided by `PubSubTemplate.getAcknowledger()`. The provided `acknowledger` allows for acknowledging or negatively acknowledging a set of `acknowledge IDs`, pertaining to a subscription. The subscription name passed to the `acknowledger` must have the following format: `project/[GCP_PROJECT_ID]/subscriptions/[SUBSCRIPTION_NAME]`.

`PubSubTemplate` uses a special `subscriber` generated by its `SubscriberFactory` to synchronously pull messages.

If the message payload contains a serialized POJO, it can be retrieved as a `Class` compatible with that serialized payload:

```
this.pubSubTemplate.getMessageConverter().fromMessage(message, MyPojo.class);
```

## 5.2 Pub/Sub management

`PubSubAdmin` is the abstraction provided by Spring Cloud GCP to manage Google Cloud Pub/Sub resources. It allows for the creation, deletion and listing of topics and subscriptions.

`PubSubAdmin` depends on `GcpProjectIdProvider` and either a `CredentialsProvider` or a `TopicAdminClient` and a `SubscriptionAdminClient`. If given a `CredentialsProvider`, it creates a `TopicAdminClient` and a `SubscriptionAdminClient` with the Google Cloud Java

Library for Pub/Sub default settings. The Spring Boot starter for GCP Pub/Sub auto-configures a `PubSubAdmin` object using the `GcpProjectIdProvider` and the `CredentialsProvider` auto-configured by the Spring Boot GCP Core starter.

## Creating a topic

`PubSubAdmin` implements a method to create topics:

```
public Topic createTopic(String topicName)
```

Here is an example of how to create a Google Cloud Pub/Sub topic:

```
public void newTopic() {
    pubSubAdmin.createTopic("topicName");
}
```

## Deleting a topic

`PubSubAdmin` implements a method to delete topics:

```
public void deleteTopic(String topicName)
```

Here is an example of how to delete a Google Cloud Pub/Sub topic:

```
public void deleteTopic() {
    pubSubAdmin.deleteTopic("topicName");
}
```

## Listing topics

`PubSubAdmin` implements a method to list topics:

```
public List<Topic> listTopics
```

Here is an example of how to list every Google Cloud Pub/Sub topic name in a project:

```
public List<String> listTopics() {
    return pubSubAdmin
        .listTopics()
        .stream()
        .map(Topic::getNameAsTopicName)
        .map(TopicName::getTopic)
        .collect(Collectors.toList());
}
```

## Creating a subscription

`PubSubAdmin` implements a method to create subscriptions to existing topics:

```
public Subscription createSubscription(String subscriptionName, String topicName, Integer ackDeadline,
    String pushEndpoint)
```

Here is an example of how to create a Google Cloud Pub/Sub subscription:

```
public void newSubscription() {
    pubSubAdmin.createSubscription("subscriptionName", "topicName", 10, "http://my.endpoint/push");
}
```

Alternative methods with default settings are provided for ease of use. The default value for `ackDeadline` is 10 seconds. If `pushEndpoint` isn't specified, the subscription uses message pulling, instead.

```
public Subscription createSubscription(String subscriptionName, String topicName)
```

```
public Subscription createSubscription(String subscriptionName, String topicName, Integer ackDeadline)
```

```
public Subscription createSubscription(String subscriptionName, String topicName, String pushEndpoint)
```

## Deleting a subscription

PubSubAdmin implements a method to delete subscriptions:

```
public void deleteSubscription(String subscriptionName)
```

Here is an example of how to delete a Google Cloud Pub/Sub subscription:

```
public void deleteSubscription() {
    pubSubAdmin.deleteSubscription("subscriptionName");
}
```

## Listing subscriptions

PubSubAdmin implements a method to list subscriptions:

```
public List<Subscription> listSubscriptions()
```

Here is an example of how to list every subscription name in a project:

```
public List<String> listSubscriptions() {
    return pubSubAdmin
        .listSubscriptions()
        .stream()
        .map(Subscription::getNameAsSubscriptionName)
        .map(SubscriptionName::getSubscription)
        .collect(Collectors.toList());
}
```

## 5.3 Configuration

The Spring Boot starter for Google Cloud Pub/Sub provides the following configuration options:

Name	Description	Required	Default value
spring.cloud.gcp.pubsub.autoconfigure.enabled	Enables or disables Pub/Sub auto-configuration	No	true
spring.cloud.gcp.pubsub.subscriber.executor-threads	Number of threads used by Subscriber instances created by SubscriberFactory	No	4
spring.cloud.gcp.pubsub.publisher.executor-threads	Number of threads used by Publisher instances created by PublisherFactory	No	4
spring.cloud.gcp.pubsub.project-id	Google project ID where the Google Cloud Pub/Sub API is hosted, if	No	

	different from the one in the <a href="#">Spring Cloud GCP Core Module</a>		
<code>spring.cloud.gcp.pubsub.credentials.location</code>	OAuth2 credentials for authenticating with the Google Cloud Pub/Sub API, if different from the ones in the <a href="#">Spring Cloud GCP Core Module</a>	No	
<code>spring.cloud.gcp.pubsub.credentials.scope</code>	OAuth2 scope for Spring Cloud GCP Pub/Sub credentials	No	<a href="https://www.googleapis.com/auth/pubsub">https://www.googleapis.com/auth/pubsub</a>
<code>spring.cloud.gcp.pubsub.pull-count</code>	The number of parallel workers	No	The available number of processors
<code>spring.cloud.gcp.pubsub.ack-extension-period</code>	The maximum period a message ack deadline will be extended, in seconds	No	0
<code>spring.cloud.gcp.pubsub.pull-endpoint</code>	The endpoint for synchronous pulling messages	No	<code>pubsub.googleapis.com:443</code>
<code>spring.cloud.gcp.pubsub.timeout-seconds</code>	TotalTimeout has ultimate control over how long the logic should keep trying the remote call until it gives up completely. The higher the total timeout, the more retries can be attempted.	No	0
<code>spring.cloud.gcp.pubsub.retry-delay-second</code>	InitialRetryDelay controls the delay before the first retry. Subsequent retries will use this value adjusted according to the <code>RetryDelayMultiplier</code> .	No	0
<code>spring.cloud.gcp.pubsub.retry-delay-multiplier</code>	<code>RetryDelayMultiplier</code> controls the change in retry delay. The retry delay of the previous call is multiplied by the <code>RetryDelayMultiplier</code>	No	1

	to calculate the retry delay for the next call.		
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code> <code>retry-delay-seconds</code>	<b>MaxRetryDelay</b> puts a limit on the value of the retry delay, so that the <code>RetryDelayMultiplier</code> can't increase the retry delay higher than this amount.	No	0
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code> <code>attempts</code>	<b>MaxAttempts</b> defines the maximum number of attempts to perform. If this value is greater than 0, and the number of attempts reaches this limit, the logic will give up retrying even if the total retry time is still lower than <code>TotalTimeout</code> .	No	0
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code>	<b> jitter</b> determines if the delay time should be randomized.	No	true
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code> <code>initial-rpc-timeout-seconds</code>	<b>InitialRpcTimeout</b> controls the timeout for the initial RPC. Subsequent calls will use this value adjusted according to the <code>RpcTimeoutMultiplier</code> .	No	0
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code> <code>timeout-multiplier</code>	<b>RpcTimeoutMultiplier</b> controls the change in RPC timeout. The timeout of the previous call is multiplied by the <code>RpcTimeoutMultiplier</code> to calculate the timeout for the next call.	No	1
<code>spring.cloud.gcp.pubsub</code> <code>[subscriber,publisher]</code> <code>rpc-timeout-seconds</code>	<b>MaxRpcTimeout</b> puts a limit on the value of the RPC timeout, so that the <code>RpcTimeoutMultiplier</code> can't increase the RPC timeout higher than this amount.	No	0

<code>spring.cloud.gcp.publisher.flow-control.max-outstanding-element-count</code>	Maximum number of outstanding elements to keep in memory before enforcing flow control.	No	unlimited
<code>spring.cloud.gcp.publisher.flow-control.max-outstanding-request-bytes</code>	Maximum number of outstanding bytes to keep in memory before enforcing flow control.	No	unlimited
<code>spring.cloud.gcp.publisher.flow-control.limit-exceeded-behavior</code>	The behavior when the specified limits are exceeded.	No	Block
<code>spring.cloud.gcp.publisher.batching.element-count-threshold</code>	The element count threshold to use for batching.	No	unset (threshold does not apply)
<code>spring.cloud.gcp.publisher.batching.request-byte-threshold</code>	The request byte threshold to use for batching.	No	unset (threshold does not apply)
<code>spring.cloud.gcp.publisher.batching.delay-threshold-seconds</code>	The delay threshold to use for batching. After this amount of time has elapsed (counting from the first element added), the elements will be wrapped up in a batch and sent.	No	unset (threshold does not apply)
<code>spring.cloud.gcp.publisher.batching.enabled</code>	Enables batching.	No	false



## 6. Spring Resources

[Spring Resources](#) are an abstraction for a number of low-level resources, such as file system files, classpath files, servlet context-relative files, etc. Spring Cloud GCP adds a new resource type: a Google Cloud Storage (GCS) object.

A Spring Boot starter is provided to auto-configure the various Storage components.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-storage</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-storage'
}
```

This starter is also available from [Spring Initializr](#) through the `GCP Storage` entry.

A [sample application](#) is available.

### 6.1 Google Cloud Storage

The Spring Resource Abstraction for Google Cloud Storage allows GCS objects to be accessed by their GCS URL using the `@Value` annotation

```
@Value("gs://[YOUR_GCS_BUCKET]/[GCS_FILE_NAME]")
private Resource gcsResource;
```

or the Spring application context

```
SpringApplication.run(...).getResource("gs://[YOUR_GCS_BUCKET]/[GCS_FILE_NAME]");
```

This creates a `Resource` object that can be used to read the object, among [other possible operations](#).

It is also possible to write to a `Resource`, although a `WritableResource` is required.

```
@Value("gs://[YOUR_GCS_BUCKET]/[GCS_FILE_NAME]")
private Resource gcsResource;
...
try (OutputStream os = ((WritableResource) gcsResource).getOutputStream()) {
    os.write("foo".getBytes());
}
```

If the resource path refers to an object on Google Cloud Storage (as opposed to a bucket), then the resource can be cast as a `GoogleStorageResourceObject` and the `getGoogleStorageObject` method can be called to obtain a `Blob`. This type represents a GCS file, which has associated [metadata](#), such as content-type, that can be set. The `createSignedUrl` method can also be used to obtain [signed URLs](#) for GCS objects. However, creating signed URLs requires that the resource was created using service account credentials.

The Spring Boot Starter for Google Cloud Storage auto-configures the `Storage` bean required by the `spring-cloud-gcp-storage` module, based on the `CredentialsProvider` provided by the Spring Boot GCP starter.

## 6.2 Configuration

The Spring Boot Starter for Google Cloud Storage provides the following configuration options:

Name	Description	Required	Default value
<code>spring.cloud.gcp.storage.create-files</code>	Creates files and buckets on Google Cloud Storage when writes are made to non-existent files	No	true
<code>spring.cloud.gcp.storage.credentials.location</code>	OAuth2 credentials for authenticating with the Google Cloud Storage API, if different from the ones in the <a href="#">Spring Cloud GCP Core Module</a>	No	
<code>spring.cloud.gcp.storage.credentials.scopes</code>	OAuth2 scope for Spring Cloud GCP Storage credentials	No	<a href="https://www.googleapis.com/auth/devstorage.read_write">https://www.googleapis.com/auth/devstorage.read_write</a>

## 7. Spring JDBC

Spring Cloud GCP adds integrations with [Spring JDBC](#) so you can run your MySQL or PostgreSQL databases in Google Cloud SQL using Spring JDBC, or other libraries that depend on it like Spring Data JPA.

The Cloud SQL support is provided by Spring Cloud GCP in the form of two Spring Boot starters, one for MySQL and another one for PostgreSQL. The role of the starters is to read configuration from properties and assume default settings so that user experience connecting to MySQL and PostgreSQL is as simple as possible.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-sql-mysql</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-sql-postgresql</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-sql-mysql'
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-sql-postgresql'
}
```

### 7.1 Prerequisites

In order to use the Spring Boot Starters for Google Cloud SQL, the Google Cloud SQL API must be enabled in your GCP project.

To do that, go to the [API library page](#) of the Google Cloud Console, search for "Cloud SQL API", click the first result and enable the API.

#### Note

There are several similar "Cloud SQL" results. You must access the "Google Cloud SQL API" one and enable the API from there.

Available sample applications:

- [Spring Cloud GCP SQL](#)
- [Spring Data JPA with Spring Cloud GCP SQL](#)

### 7.2 Spring Boot Starter for Google Cloud SQL

The Spring Boot Starters for Google Cloud SQL provide an auto-configured [DataSource](#) object. Coupled with Spring JDBC, it provides a [JdbcTemplate](#) object bean that allows for operations such as querying and modifying a database.

```
public List<Map<String, Object>> listUsers() {
    return jdbcTemplate.queryForList("SELECT * FROM user;");
}
```

You can rely on [Spring Boot data source auto-configuration](#) to configure a `DataSource` bean. In other words, properties like the SQL username, `spring.datasource.username`, and password, `spring.datasource.password` can be used. There is also some configuration specific to Google Cloud SQL:

Property name	Description	Default value	Unused if specified property(ies)
<code>spring.cloud.gcp.sql.enabled</code>	Enables or disables Cloud SQL auto configuration	true	
<code>spring.cloud.gcp.sql.database-name</code>	Name of the database to connect to.		<code>spring.datasource.url</code>
<code>spring.cloud.gcp.sql.connection-name</code>	A string containing a Google Cloud SQL instance's project ID, region and name, each separated by a colon. For example, <code>my-project-id:my-region:my-instance-name</code> .		<code>spring.datasource.url</code>
<code>spring.cloud.gcp.sql.credentials-path</code>	File system path to the Google OAuth2 credentials private key file. Used to authenticate and authorize new connections to a Google Cloud SQL instance.	Default credentials provided by the Spring GCP Boot starter	

## DataSource creation flow

Based on the previous properties, the Spring Boot starter for Google Cloud SQL creates a `CloudSqlJdbcInfoProvider` object which is used to obtain an instance's JDBC URL and driver class name. If you provide your own `CloudSqlJdbcInfoProvider` bean, it is used instead and the properties related to building the JDBC URL or driver class are ignored.

The `DataSourceProperties` object provided by Spring Boot Autoconfigure is mutated in order to use the JDBC URL and driver class names provided by `CloudSqlJdbcInfoProvider`, unless those values were provided in the properties. It is in the `DataSourceProperties` mutation step that the credentials factory is registered in a system property to be `SqlCredentialFactory`.

`DataSource` creation is delegated to [Spring Boot](#). You can select the type of connection pool (e.g., Tomcat, HikariCP, etc.) by [adding their dependency to the classpath](#).

Using the created `DataSource` in conjunction with Spring JDBC provides you with a fully configured and operational `JdbcTemplate` object that you can use to interact with your SQL database. You can connect to your database with as little as a database and instance names.

## Troubleshooting tips

### Connection issues

If you're not able to connect to a database and see an endless loop of `Connecting to Cloud SQL instance [...] on IP [...]`, it's likely that exceptions are being thrown and logged at a level lower than your logger's level. This may be the case with HikariCP, if your logger is set to INFO or higher level.

To see what's going on in the background, you should add a `logback.xml` file to your application resources folder, that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <include resource="org/springframework/boot/logging/logback/base.xml"/>
  <logger name="com.zaxxer.hikari.pool" level="DEBUG"/>
</configuration>
```

**Errors like `c.g.cloud.sql.core.SslSocketFactory : Re-throwing cached exception due to attempt to refresh instance information too soon after error`**

If you see a lot of errors like this in a loop and can't connect to your database, this is usually a symptom that something isn't right with the permissions of your credentials or the Google Cloud SQL API is not enabled. Verify that the Google Cloud SQL API is enabled in the Cloud Console and that your service account has the [necessary IAM roles](#).

To find out what's causing the issue, you can enable DEBUG logging level as mentioned [above](#).

### PostgreSQL: `java.net.SocketException: already connected` issue

We found this exception to be common if your Maven project's parent is `spring-boot` version `1.5.x`, or in any other circumstance that would cause the version of the `org.postgresql:postgresql` dependency to be an older one (e.g., `9.4.1212.jre7`).

To fix this, re-declare the dependency in its correct version. For example, in Maven:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.1.1</version>
</dependency>
```

## 8. Spring Integration

Spring Cloud GCP provides Spring Integration adapters that allow your applications to use Enterprise Integration Patterns backed up by Google Cloud Platform services.

### 8.1 Channel Adapters for Google Cloud Pub/Sub

The channel adapters for Google Cloud Pub/Sub connect your Spring [MessageChannels](#) to Google Cloud Pub/Sub topics and subscriptions. This enables messaging between different processes, applications or micro-services backed up by Google Cloud Pub/Sub.

The Spring Integration Channel Adapters for Google Cloud Pub/Sub are included in the `spring-cloud-gcp-pubsub` module.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-pubsub</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.integration</groupId>
  <artifactId>spring-integration-core</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-pubsub'
    compile group: 'org.springframework.integration', name: 'spring-integration-core'
}
```

A [sample application](#) is available.

### Inbound channel adapter

`PubSubInboundChannelAdapter` is the inbound channel adapter for GCP Pub/Sub that listens to a GCP Pub/Sub subscription for new messages. It converts new messages to an internal Spring [Message](#) and then sends it to the bound output channel.

Google Pub/Sub treats message payloads as byte arrays. So, by default, the inbound channel adapter will construct the Spring `Message` with `byte[]` as the payload. However, you can change the desired payload type by setting the `payloadType` property of the `PubSubInboundChannelAdapter`. The `PubSubInboundChannelAdapter` delegates to conversion to the desired payload type to the `PubSubMessageConverter` configured in the `PubSubTemplate`.

To use the inbound channel adapter, a `PubSubInboundChannelAdapter` must be provided and configured on the user application side.

```
@Bean
public MessageChannel pubsubInputChannel() {
    return new PublishSubscribeChannel();
}

@Bean
public PubSubInboundChannelAdapter messageChannelAdapter(
    @Qualifier("pubsubInputChannel") MessageChannel inputChannel,
    SubscriberFactory subscriberFactory) {
    PubSubInboundChannelAdapter adapter =
```

```

        new PubSubInboundChannelAdapter(subscriberFactory, "subscriptionName");
        adapter.setOutputChannel(inputChannel);
        adapter.setAckMode(AckMode.MANUAL);

        return adapter;
    }

```

In the example, we first specify the `MessageChannel` where the adapter is going to write incoming messages to. The `MessageChannel` implementation isn't important here. Depending on your use case, you might want to use a `MessageChannel` other than `PublishSubscribeChannel`.

Then, we declare a `PubSubInboundChannelAdapter` bean. It requires the channel we just created and a `SubscriberFactory`, which creates `Subscriber` objects from the Google Cloud Java Client for Pub/Sub. The Spring Boot starter for GCP Pub/Sub provides a configured `SubscriberFactory`.

It is also possible to set the message acknowledgement mode on the adapter, which is automatic by default. On automatic acking, a message is acked with GCP Pub/Sub if the adapter sent it to the channel and no exceptions were thrown. If a `RuntimeException` is thrown while the message is processed, then the message is nacked. On manual acking, the adapter attaches an `AckReplyConsumer` object to the `Message` headers, which users can extract using the `GcpPubSubHeaders.ACKNOWLEDGEMENT` key and use to (n)ack a message.

```

@Bean
@ServiceActivator(inputChannel = "pubsubInputChannel")
public MessageHandler messageReceiver() {
    return message -> {
        LOGGER.info("Message arrived! Payload: " + new String((byte[]) message.getPayload()));
        AckReplyConsumer consumer =
            message.getHeaders().get(GcpPubSubHeaders.ACKNOWLEDGEMENT, AckReplyConsumer.class);
        consumer.ack();
    };
}

```

## Outbound channel adapter

`PubSubMessageHandler` is the outbound channel adapter for GCP Pub/Sub that listens for new messages on a Spring `MessageChannel`. It uses `PubSubTemplate` to post them to a GCP Pub/Sub topic.

To construct a Pub/Sub representation of the message, the outbound channel adapter needs to convert the Spring `Message` payload to a byte array representation expected by Pub/Sub. It delegates this conversion to the `PubSubTemplate`. To customize the conversion, you can specify a `PubSubMessageConverter` in the `PubSubTemplate` that should convert the `Object` payload and headers of the Spring `Message` to a `PubsubMessage`.

To use the outbound channel adapter, a `PubSubMessageHandler` bean must be provided and configured on the user application side.

```

@Bean
@ServiceActivator(inputChannel = "pubsubOutputChannel")
public MessageHandler messageSender(PubSubTemplate pubsubTemplate) {
    return new PubSubMessageHandler(pubsubTemplate, "topicName");
}

```

The provided `PubSubTemplate` contains all the necessary configuration to publish messages to a GCP Pub/Sub topic.

`PubSubMessageHandler` publishes messages asynchronously by default. A publish timeout can be configured for synchronous publishing. If none is provided, the adapter waits indefinitely for a response.

It is possible to set user-defined callbacks for the `publish()` call in `PubSubMessageHandler` through the `setPublishFutureCallback()` method. These are useful to process the message ID, in case of success, or the error if any was thrown.

To override the default destination you can use the `GcpPubSubHeaders.DESTINATION` header.

```
@Autowired
private MessageChannel pubsubOutputChannel;

public void handleMessage(Message<?> msg) throws MessagingException {
    final Message<?> message = MessageBuilder
        .withPayload(msg.getPayload())
        .setHeader(GcpPubSubHeaders.TOPIC, "customTopic").build();
    pubsubOutputChannel.send(message);
}
```

It is also possible to set an SpEL expression for the topic with the `setTopicExpression()` or `setTopicExpressionString()` methods.

## Header mapping

These channel adapters contain header mappers that allow you to map, or filter out, headers from Spring to Google Cloud Pub/Sub messages, and vice-versa. By default, the inbound channel adapter maps every header on the Google Cloud Pub/Sub messages to the Spring messages produced by the adapter. The outbound channel adapter maps every header from Spring messages into Google Cloud Pub/Sub ones, except the ones added by Spring, like headers with key `"id"`, `"timestamp"` and `"gcp_pubsub_acknowledgement"`. In the process, the outbound mapper also converts the value of the headers into string.

Each adapter declares a `setHeaderMapper()` method to let you further customize which headers you want to map from Spring to Google Cloud Pub/Sub, and vice-versa.

For example, to filter out headers `"foo"`, `"bar"` and all headers starting with the prefix `"prefix_"`, you can use `setHeaderMapper()` along with the `PubSubHeaderMapper` implementation provided by this module.

```
PubSubMessageHandler adapter = ...
...
PubSubHeaderMapper headerMapper = new PubSubHeaderMapper();
headerMapper.setOutboundHeaderPatterns("!foo", "!bar", "!prefix_**", "**");
adapter.setHeaderMapper(headerMapper);
```

### Note

The order in which the patterns are declared in `PubSubHeaderMapper.setOutboundHeaderPatterns()` and `PubSubHeaderMapper.setInboundHeaderPatterns()` matters. The first patterns have precedence over the following ones.

In the previous example, the `"**"` pattern means every header is mapped. However, because it comes last in the list, [the previous patterns take precedence](#).

## 8.2 Channel Adapters for Google Cloud Storage

The channel adapters for Google Cloud Storage allow you to read and write files to Google Cloud Storage through `MessageChannels`.



Spring Cloud GCP provides two inbound adapters, `GcsInboundFileSynchronizingMessageSource` and `GcsStreamingMessageSource`, and one outbound adapter, `GcsMessageHandler`.

The Spring Integration Channel Adapters for Google Cloud Storage are included in the `spring-cloud-gcp-storage` module.

To use the Storage portion of Spring Integration for Spring Cloud GCP, you must also provide the `spring-integration-file` dependency, since they aren't pulled transitively.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-storage</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.integration</groupId>
  <artifactId>spring-integration-file</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-storage'
    compile group: 'org.springframework.integration', name: 'spring-integration-file'
}
```

A [sample application](#) is available.

## Inbound channel adapter

The Google Cloud Storage inbound channel adapter polls a Google Cloud Storage bucket for new files and sends each of them in a `Message` payload to the `MessageChannel` specified in the `@InboundChannelAdapter` annotation. The files are temporarily stored in a folder in the local file system.

Here is an example of how to configure a Google Cloud Storage inbound channel adapter.

```
@Bean
@InboundChannelAdapter(channel = "new-file-channel", poller = @Poller(fixedDelay = "5000"))
public MessageSource<File> synchronizerAdapter(Storage gcs) {
    GcsInboundFileSynchronizer synchronizer = new GcsInboundFileSynchronizer(gcs);
    synchronizer.setRemoteDirectory("your-gcs-bucket");

    GcsInboundFileSynchronizingMessageSource synchAdapter =
        new GcsInboundFileSynchronizingMessageSource(synchronizer);
    synchAdapter.setLocalDirectory(new File("local-directory"));

    return synchAdapter;
}
```

## Inbound streaming channel adapter

The inbound streaming channel adapter is similar to the normal inbound channel adapter, except it does not require files to be stored in the file system.

Here is an example of how to configure a Google Cloud Storage inbound streaming channel adapter.

```
@Bean
@InboundChannelAdapter(channel = "streaming-channel", poller = @Poller(fixedDelay = "5000"))
```

```
public MessageSource<InputStream> streamingAdapter(Storage gcs) {
    GcsStreamingMessageSource adapter =
        new GcsStreamingMessageSource(new GcsRemoteFileTemplate(new GcsSessionFactory(gcs)));
    adapter.setRemoteDirectory("your-gcs-bucket");
    return adapter;
}
```

## Outbound channel adapter

The outbound channel adapter allows files to be written to Google Cloud Storage. When it receives a `Message` containing a payload of type `File`, it writes that file to the Google Cloud Storage bucket specified in the adapter.

Here is an example of how to configure a Google Cloud Storage outbound channel adapter.

```
@Bean
@ServiceActivator(inputChannel = "writeFiles")
public MessageHandler outboundChannelAdapter(Storage gcs) {
    GcsMessageHandler outboundChannelAdapter = new GcsMessageHandler(new GcsSessionFactory(gcs));
    outboundChannelAdapter.setRemoteDirectoryExpression(new ValueExpression<>("your-gcs-bucket"));

    return outboundChannelAdapter;
}
```

## 9. Spring Cloud Stream

Spring Cloud GCP provides a [Spring Cloud Stream](#) binder to Google Cloud Pub/Sub.

The provided binder relies on the [Spring Integration Channel Adapters for Google Cloud Pub/Sub](#).

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-pubsub-stream-binder</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-pubsub-stream-binder'
}
```

A [sample application](#) is available.

### 9.1 Overview

This binder binds producers to Google Cloud Pub/Sub topics and consumers to subscriptions.

#### Note

Partitioning and consumer groups are not currently supported by this binder.

### 9.2 Configuration

You can configure the Spring Cloud Stream Binder for Google Cloud Pub/Sub to automatically generate the underlying resources, like the Google Cloud Pub/Sub subscriptions for the consumers. For that, you can use the `spring.cloud.stream.gcp.pubsub.bindings.[CHANNEL-NAME].consumer.auto-create-resources` property, which is turned ON by default.

If automatic resource creation is turned ON and the subscription and the topic do not exist for a consumer, a subscription and a topic will be created with the same name. For example, for the following configuration, a topic and a subscription called `myConsumer` would be created.

**application.properties.**

```
spring.cloud.stream.bindings.output.destination=myConsumer

spring.cloud.stream.gcp.pubsub.bindings.output.consumer.auto-create-resources=true
```

If you are using Pub/Sub auto-configuration from the Spring Cloud GCP Pub/Sub Starter, you should refer to the [configuration](#) section for other Pub/Sub parameters.

#### Note

To use this binder with a [running emulator](#), configure its host and port via `spring.cloud.gcp.pubsub.emulator-host`.

## 10. Spring Cloud Sleuth

[Spring Cloud Sleuth](#) is an instrumentation framework for Spring Boot applications. It captures trace informations and can forward traces to services like Zipkin for storage and analysis.

Google Cloud Platform provides its own managed distributed tracing service called [Stackdriver Trace](#). Instead of running and maintaining your own Zipkin instance and storage, you can use Stackdriver Trace to store traces, view trace details, generate latency distributions graphs, and generate performance regression reports.

This Spring Cloud GCP starter can forward Spring Cloud Sleuth traces to Stackdriver Trace without an intermediary Zipkin server.

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-trace</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-trace'
}
```

You must enable Stackdriver Trace API from the Google Cloud Console in order to capture traces. Navigate to the [Stackdriver Trace API](#) for your project and make sure it's enabled.

A [sample application](#) is available.

### Note

If you are already using a Zipkin server capturing trace information from multiple platform/frameworks, you also use a [Stackdriver Zipkin proxy](#) to forward those traces to Stackdriver Trace without modifying existing applications.

### 10.1 Tracing

Spring Cloud Sleuth uses the [Brave tracer](#) to generate traces. This integration enables Brave to use the [StackdriverTracePropagation](#) propagation.

A propagation is responsible for extracting trace context from an entity (e.g., an HTTP servlet request) and for injecting trace context into an entity. A canonical example of the propagation usage is a web server that receives an HTTP request, which triggers other HTTP requests from the server before returning an HTTP response to the original caller. In the case of `StackdriverTracePropagation`, first it looks for trace context in the `x-cloud-trace-context` key (e.g., an HTTP request header). The value of the `x-cloud-trace-context` key can be formatted in three different ways:

- `x-cloud-trace-context: TRACE_ID`
- `x-cloud-trace-context: TRACE_ID/SPAN_ID`
- `x-cloud-trace-context: TRACE_ID/SPAN_ID;o=TRACE_TRUE`

TRACE\_ID is a 32-character hexadecimal value that encodes a 128-bit number.

SPAN\_ID is an unsigned long. Since Stackdriver Trace doesn't support span joins, a new span ID is always generated, regardless of the one specified in x-cloud-trace-context.

TRACE\_TRUE can either be 0 if the entity should be untraced, or 1 if it should be traced. However, at the moment, if TRACE\_TRUE is set to 1, the entity isn't necessarily traced. Currently, to make sure a request is traced, the Sleuth property `spring.sleuth.sampler.probability=1` should be used, to trace every entity.

If a x-cloud-trace-context key isn't found, StackdriverTracePropagation falls back to tracing with the [X-B3 headers](#).

## 10.2 Spring Boot Starter for Stackdriver Trace

Spring Boot Starter for Stackdriver Trace uses Spring Cloud Sleuth and auto-configures a [StackdriverSender](#) that sends the Sleuth's trace information to Stackdriver Trace.

All configurations are optional:

Name	Description	Required	Default value
<code>spring.cloud.gcp.trace.autoconfigure</code>	Auto-configure Spring Cloud Sleuth to send traces to Stackdriver Trace.	No	true
<code>spring.cloud.gcp.trace.project-id</code>	Overrides the project ID from the <a href="#">Spring Cloud GCP Module</a>	No	
<code>spring.cloud.gcp.trace.credentials.location</code>	Overrides the credentials location from the <a href="#">Spring Cloud GCP Module</a>	No	
<code>spring.cloud.gcp.trace.credentials.scopes</code>	Overrides the credentials scopes from the <a href="#">Spring Cloud GCP Module</a>	No	
<code>spring.cloud.gcp.trace.executor-threads</code>	Number of threads used by the Trace executor	No	4
<code>spring.cloud.gcp.trace.http2-authority</code>	HTTP/2 authority the channel claims to be connecting to.	No	
<code>spring.cloud.gcp.trace.compression</code>	Name of the compression to use in Trace calls	No	
<code>spring.cloud.gcp.trace.call-deadline-ms</code>	Call deadline in milliseconds	No	

<code>spring.cloud.gcp.trace.inbound-size</code>	Maximum size for inbound messages	No	
<code>spring.cloud.gcp.trace.outbound-size</code>	Maximum size for outbound messages	No	
<code>spring.cloud.gcp.trace.for-ready</code>	<a href="#">Waits for the channel to be ready</a> in case of a transient failure	No	false

You can use core Spring Cloud Sleuth properties to control Sleuth's sampling rate, etc. Read [Sleuth documentation](#) for more information on Sleuth configurations.

For example, when you are testing to see the traces are going through, you can set the sampling rate to 100%.

```
spring.sleuth.sampler.probability=1 # Send 100% of the request traces to Stackdriver.
spring.sleuth.web.skipPattern=(^cleanup.*|.+favicon.*) # Ignore some URL paths.
```

Spring Cloud GCP Trace does override some Sleuth configurations:

- Always uses 128-bit Trace IDs. This is required by Stackdriver Trace.
- Does not use Span joins. Span joins will share the span ID between the client and server Spans. Stackdriver requires that every Span ID within a Trace to be unique, so Span joins are not supported.
- Uses `StackdriverHttpClientParser` and `StackdriverHttpServerParser` by default to populate Stackdriver related fields.

## 10.3 Integration with Logging

Integration with Stackdriver Logging is available through the [Stackdriver Logging Support](#). If the Trace integration is used together with the Logging one, the request logs will be associated to the corresponding traces. The trace logs can be viewed by going to the [Google Cloud Console Trace List](#), selecting a trace and pressing the `Logs` # `View` link in the `Details` section.

## 11. Stackdriver Logging Support

Maven coordinates, using Spring Cloud GCP BOM:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-logging</artifactId>
</dependency>
```

Gradle coordinates:

```
dependencies {
    compile group: 'org.springframework.cloud', name: 'spring-cloud-gcp-starter-logging'
}
```

[Stackdriver Logging](#) is the managed logging service provided by Google Cloud Platform.

This module provides support for associating a web request trace ID with the corresponding log entries. It does so by retrieving the X-B3-TraceId value from the [Mapped Diagnostic Context \(MDC\)](#), which is set by Spring Cloud Sleuth. If Spring Cloud Sleuth isn't used, the configured `TraceIdExtractor` extracts the desired header value and sets it as the log entry's trace ID. This allows grouping of log messages by request, for example, in the [Google Cloud Console Logs viewer](#).

### Note

Due to the way logging is set up, the GCP project ID and credentials defined in `application.properties` are ignored. Instead, you should set the `GOOGLE_CLOUD_PROJECT` and `GOOGLE_APPLICATION_CREDENTIALS` environment variables to the project ID and credentials private key location, respectively. You can do this easily if you're using the [Google Cloud SDK](#), using the `gcloud config set project [YOUR_PROJECT_ID]` and `gcloud auth application-default login` commands, respectively.

A [sample application](#) is available.

### 11.1 Web MVC Interceptor

For use in Web MVC-based applications, `TraceIdLoggingWebMvcInterceptor` is provided that extracts the request trace ID from an HTTP request using a `TraceIdExtractor` and stores it in a thread-local, which can then be used in a logging appender to add the trace ID metadata to log messages.

### Warning

If Spring Cloud GCP Trace is enabled, the logging module disables itself and delegates log correlation to Spring Cloud Sleuth.

`LoggingWebMvcConfigurer` configuration class is also provided to help register the `TraceIdLoggingWebMvcInterceptor` in Spring MVC applications.

Applications hosted on the Google Cloud Platform include trace IDs under the `x-cloud-trace-context` header, which will be included in log entries. However, if Sleuth is used the trace ID will be picked up from the MDC.

## 11.2 Logback Support

Currently, only Logback is supported and there are 2 possibilities to log to Stackdriver via this library with Logback: via direct API calls and through JSON-formatted console logs.

### Log via API

A Stackdriver appender is available using `org/springframework/cloud/gcp/autoconfigure/logging/logback-appender.xml`. This appender builds a Stackdriver Logging log entry from a JUL or Logback log entry, adds a trace ID to it and sends it to Stackdriver Logging.

`STACKDRIVER_LOG_NAME` and `STACKDRIVER_LOG_FLUSH_LEVEL` environment variables can be used to customize the `STACKDRIVER` appender.

### Log via Console

For Logback, a `org/springframework/cloud/gcp/autoconfigure/logging/logback-json-appender.xml` file is made available for import to make it easier to configure the JSON Logback appender.

Your configuration may then look something like this:

```
<configuration>
  <include resource="org/springframework/cloud/gcp/autoconfigure/logging/logback-json-appender.xml" />

  <root level="INFO">
    <appender-ref ref="CONSOLE_JSON" />
  </root>
</configuration>
```

If your application is running on Google Container Engine, Google Compute Engine or Google App Engine Flexible, your console logging is automatically saved to Google Stackdriver Logging. Therefore, you can just include `org/springframework/cloud/gcp/autoconfigure/logging/logback-json-appender.xml` in your logging configuration, which logs JSON entries to the console. The trace id will be set correctly.

Your Logback configuration may then look something like this:

```
<configuration>
  <include resource="org/springframework/cloud/gcp/autoconfigure/logging/logback-appender.xml" />

  <root level="INFO">
    <appender-ref ref="CONSOLE_JSON" />
  </root>
</configuration>
```

If you want to have more control over the log output, you can also configure the `ConsoleAppender` yourself. The following properties are available:

Property	Default Value	Description
<code>projectId</code>	If not set, default value is determined in the following order:  1. <code>SPRING_CLOUD_GCP_LOGGING_PROJECT_ID</code> Environmental Variable.	This is used to generate fully qualified Stackdriver Trace ID format: <code>projects/[PROJECT-ID]/traces/[TRACE-ID]</code> .  This format is required to correlate trace between



Property	Default Value	Description
	2. Value of <code>DefaultGcpProjectIdProvider.getProjectId()</code>	Stackdriver Trace and Stackdriver Logging. If <code>projectId</code> is not set and cannot be determined, then it'll log <code>traceId</code> without the fully qualified format.
<code>includeTraceId</code>	<code>true</code>	Should the <code>traceId</code> be included
<code>includeSpanId</code>	<code>true</code>	Should the <code>spanId</code> be included
<code>includeLevel</code>	<code>true</code>	Should the severity be included
<code>includeThreadName</code>	<code>true</code>	Should the thread name be included
<code>includeMDC</code>	<code>true</code>	Should all MDC properties be included. The MDC properties <code>X-B3-TraceId</code> , <code>X-B3-SpanId</code> and <code>X-Span-Export</code> provided by Spring Sleuth will get excluded as they get handled separately
<code>includeLoggerName</code>	<code>true</code>	Should the name of the logger be included
<code>includeFormattedMessage</code>	<code>true</code>	Should the formatted log message be included.
<code>includeExceptionInMessage</code>	<code>true</code>	Should the stacktrace be appended to the formatted log message. This setting is only evaluated if <code>includeFormattedMessage</code> is <code>true</code>
<code>includeContextName</code>	<code>true</code>	Should the logging context be included
<code>includeMessage</code>	<code>false</code>	Should the log message with blank placeholders be included
<code>includeException</code>	<code>false</code>	Should the stacktrace be included as a own field

This is an example of such an Logback configuration:

```
<configuration >
  <property name="projectId" value="${projectId:-${GOOGLE_CLOUD_PROJECT}}"/>

  <appender name="CONSOLE_JSON" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
```

```
<layout class="org.springframework.cloud.gcp.logging.StackdriverJsonLayout">
  <projectId>${projectId}</projectId>

  <!--<includeTraceId>true</includeTraceId>-->
  <!--<includeSpanId>true</includeSpanId>-->
  <!--<includeLevel>true</includeLevel>-->
  <!--<includeThreadName>true</includeThreadName>-->
  <!--<includeMDC>true</includeMDC>-->
  <!--<includeLoggerName>true</includeLoggerName>-->
  <!--<includeFormattedMessage>true</includeFormattedMessage>-->
  <!--<includeExceptionInMessage>true</includeExceptionInMessage>-->
  <!--<includeContextName>true</includeContextName>-->
  <!--<includeMessage>false</includeMessage>-->
  <!--<includeException>false</includeException>-->
</layout>
</encoder>
</appender>
</configuration>
```

## 12. Cloud Foundry

Spring Cloud GCP provides support for Cloud Foundry's [GCP Service Broker](#). Our Pub/Sub, Storage, Stackdriver Trace and Cloud SQL MySQL and PostgreSQL starters are Cloud Foundry aware and retrieve properties like project ID, credentials, etc., that are used in auto configuration from the Cloud Foundry environment.

In cases like Pub/Sub's topic and subscription, or Storage's bucket name, where those parameters are not used in auto configuration, you can fetch them using the VCAP mapping provided by Spring Boot. For example, to retrieve the provisioned Pub/Sub topic, you can use the `vcap.services.mypubsub.credentials.topic_name` property from the application environment.

### Note

If the same service is bound to the same application more than once, the auto configuration will not be able to choose among bindings and will not be activated for that service. This includes both MySQL and PostgreSQL bindings to the same app.

### Warning

In order for the Cloud SQL integration to work in Cloud Foundry, auto-reconfiguration must be disabled. You can do so using the `cf set-env <APP> JBP_CONFIG_SPRING_AUTO_RECONFIGURATION '{enabled: false}'` command. Otherwise, Cloud Foundry will produce a `DataSource` with an invalid JDBC URL (i.e., `jdbc:mysql://null/null`).