



Building commercetools customizations using Subscriptions and the Google Cloud Platform

Table of Contents

Introduction	3
What are Subscriptions?	3
Implementation Example	4
Use Case	4
Problem	4
Solution	5
Implementation Steps	5
Step A: Set up a GCP Project.	6
Set up a Google Cloud Account	6
Set up a Google Cloud Project	6
Install the Google Cloud SDK	6
Authenticate using Google Cloud SDK	6
Step B: Set up GCP Cloud Firestore.	6
Step C: Create a GCP Cloud Function.	8
Step D: Configure and Deploy the Cloud Function.	11
Step E: Set up the GCP Pub/Sub Topic.	13
Step F: Create a commercetools project.	14
Step G: Load commercetools sample data.	14
Step H: Set up commercetools Subscription.	15
Step I: Test commercetools/GCP Integration.	18
Step J: Provide you with resources if you need help.	20
Additional Help	21



Introduction

“By using Event Messages you can easily decouple senders and receivers both in terms of identity (you broadcast events without caring who responds to them) and time (events can be queued and forwarded when the receiver is ready to process them). Such architectures offer a great deal for scalability and modifiability due to this loose coupling.”

– [Focusing on Events](#), Martin Fowler

commercetools is a dynamically extensible, cloud-native commerce solution. It allows retailers to sculpt a solution that fits their unique needs today, and is flexible to support their evolving business strategy tomorrow.

There are many powerful extensibility features built into commercetools that handle a wide variety of use cases. For an overview of them, see [Building commercetools customizations - Overview](#).

In this whitepaper we will do a deep dive on one powerful technique for customizing commercetools: Subscriptions.

What are Subscriptions?

[Subscriptions](#) allow you to trigger custom asynchronous background processing in response to an event on the commercetools platform.

“Subscriptions allow you to be notified of new messages or changes via a Message Queue of your choice.”

– [Subscriptions](#), Platform Documentation, commercetools

Because Subscriptions execute asynchronously based on events emitted from the platform, they allow your custom solutions to be loosely coupled to commercetools. This greatly reduces the risk of your code impacting commercetools API execution and performance.

commercetools differentiates between **messages** and **changes**. A single subscription can listen to both depending on the resource. Changes are straightforward: events are fired whenever the [subscribed resource type](#) is created, updated or deleted. Messages are more specific:

