In distributed systems, such as a network of Compute Engine instances, it is challenging to reliably schedule tasks, because any individual instance might become unavailable due to autoscaling or network partitioning.

By using Cloud Scheduler for scheduling and Pub/Sub for distributed messaging, you can build an application to reliably schedule tasks across a fleet of Compute Engine instances. If you need to schedule and orchestrate complex workflows across other products or clouds, consider using Cloud Composer (/composer/) instead.

This three-part article includes the following:

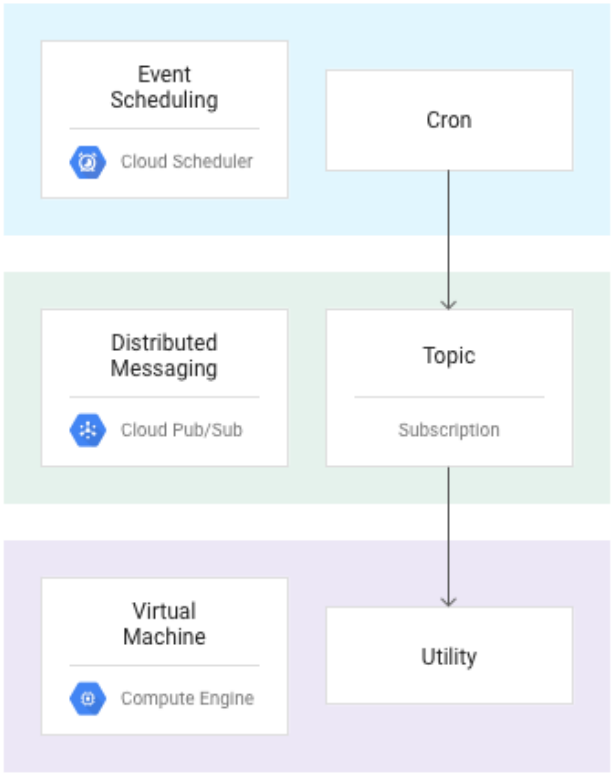
- A design pattern for the solution (#schedule-compute-engine)
- A sample implementation of the design pattern (#sample-implementation)
- Ideas for building a production-ready version of the solution (#best-practices)

Cron is the standard tool for scheduling recurring tasks on Unix systems. As the systems you build increase in complexity and become distributed, a single computer running cron can become a critical point of failure. The instance may stop due to autoscaling, or its network segment could be partitioned from systems it needs to communicate with.

Cloud Scheduler (/scheduler/) provides a fully managed, enterprise-grade service that lets you schedule events. After you have scheduled a job, Cloud Scheduler will call the configured event handlers, which can be App Engine (/appengine/) services, HTTP endpoints, or Pub/Sub (/pubsub/overview) subscriptions.

To run tasks on your Compute Engine instance in response to Cloud Scheduler events, you need to relay the events to those instances. One way to do this is by calling an HTTP endpoint that runs on your Compute Engine instances. Another option is to pass messages from Cloud Scheduler to your Compute Engine instances using Pub/Sub. This sample illustrates the second design pattern.

The following diagram provides an architectural overview of this design pattern.



In this implementation, you schedule events in Cloud Scheduler, then transmit those events to Compute Engine instances using Pub/Sub.

A utility service on your Compute Engine instances subscribes to Pub/Sub topics and runs cron jobs in response to the events it pulls down from those topics. The utility runs standard scripts; you do not need to modify your current cron scripts to use them in this sample.

By using Pub/Sub to decouple the task-scheduling logic from the logic running the commands on Compute Engine, you can update your cron scripts as needed, without updating the Cloud Scheduler configuration. You can also change your task schedule without updating the utility service on your Compute Engine instances.

Because cron jobs are typically few in number and run on an hourly, weekly, or daily schedule, this design pattern should not exceed Cloud Scheduler quotas (/scheduler/quotas), which allow tens of requests per minute and thousands per day. If it does, consider other application patterns, such as managing the timing of tasks directly in application code.

You can try out the sample implementation of this design pattern at no cost with the Google Cloud Free Tier (/free) if you aren't using those resources for other applications. If your free quotas are used by other applications in your project, the costs will be determined by your total usage of Compute Engine, Cloud Scheduler, and Pub/Sub resources.

Pricing for Cloud Scheduler (/scheduler/pricing) is based on number of jobs scheduled. Pricing for Compute Engine (/compute/pricing) is based on the type and duration of the instances used. Pricing for Pub/Sub (/pubsub/pricing) is based on the volume of data sent.

For example, if you run the sample implementation in the following section for an hour and then delete the Google Cloud resources, the cost will be approximately 1 cent. For a breakdown of the costs in this estimate, and to calculate costs for your own use case, see the Pricing calculator (https://cloud.google.com/products/calculator/#id=beb5326f-90c3-4842-9c3f-a3761b40fbe3).

Note: These costs are only estimates; Google Cloud pricing is subject to change.

A sample implementation of this design pattern, Sample: Reliable task scheduling on Compute Engine (https://github.com/GoogleCloudPlatform/reliable-task-scheduling-compute-engine-sample), is available on GitHub.

The sample consists of two parts:

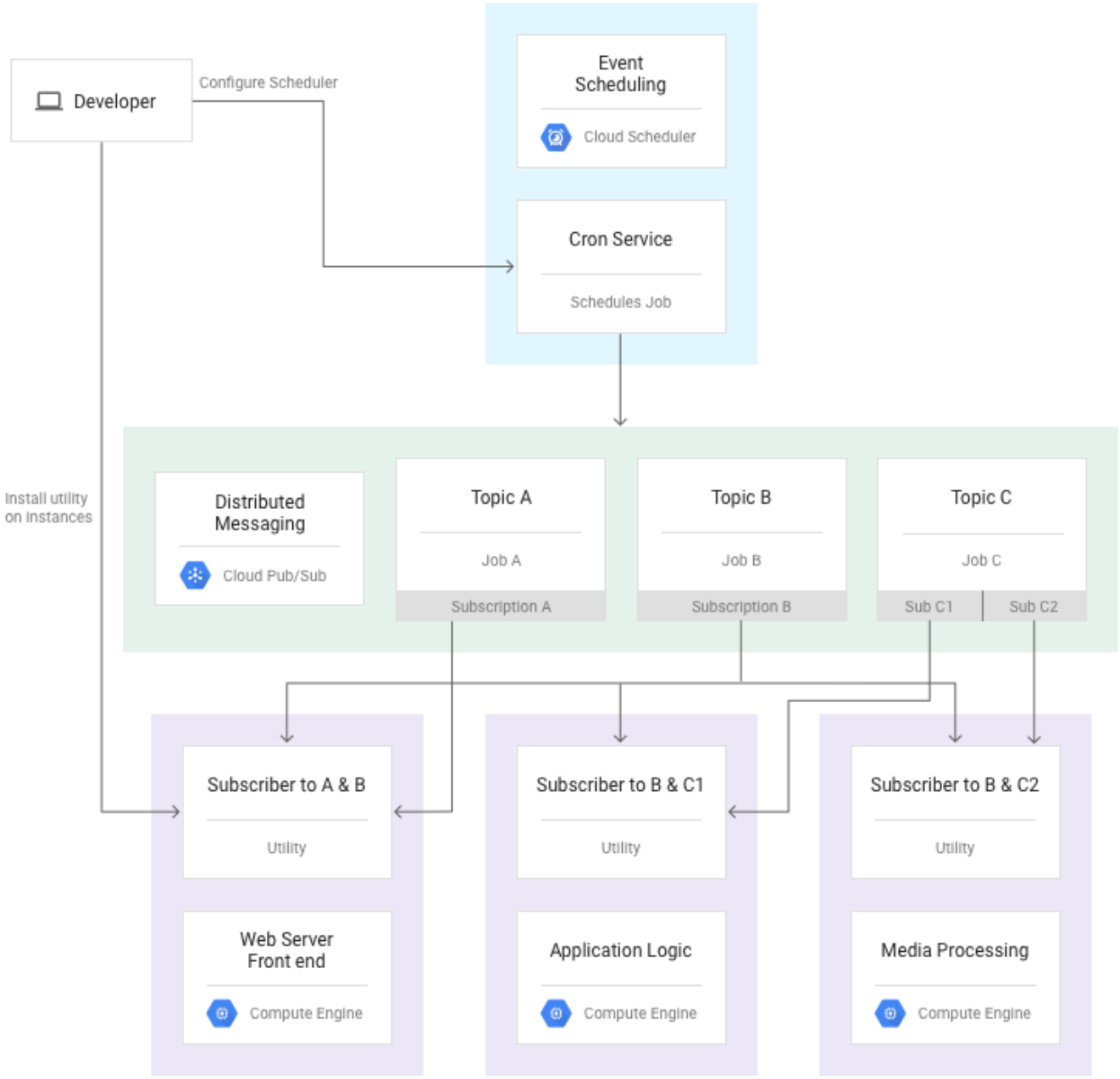
- Instructions for configuring Cloud Scheduler and Pub/Sub.
- A utility that runs on Compute Engine. This utility monitors a Pub/Sub topic. When it detects a new message, it runs the corresponding command locally on the server.

The readme file included with the sample describes the sample in further detail, as well as how to run the sample code on Google Cloud.

For the specific case of starting and stopping instances on a schedule, see Scheduling Compute Instances with Cloud Scheduler (/scheduler/docs/start-and-stop-compute-engine-instances-on-a-schedule).

The sample illustrates one way to implement a reliable scheduling solution for Compute Engine using Cloud Scheduler. It's a useful design pattern because it separates the scheduling logic from the logic that runs commands on the Compute Engine instance, making it possible to change the location and execution of your tasks without having to update the scheduling logic.

The following diagram shows the flow of messages in this sample. By specifying which instances subscribe to a given topic, you can control whether a cron job runs on a single instance or several instances.



Another advantage of this architecture is the control it gives you over how cron jobs are routed to your instances.

You can send different cron messages to different sets of servers, as illustrated by Pub/Sub topics A and C. The tasks in Topic A are sent to a single subscriber, whereas several servers subscribe to Topic C. You might use this strategy to run one set of commands on your web server and another set on your other servers.

Another option is to run a command on one of several servers. This is illustrated by topic B. In this case, multiple servers share a single subscription. Messages published to topic B are

handled by the first server to claim that message, and the corresponding command runs only on that server. You might use this approach to perform nightly data analysis that needs to run on only a single server.

You can modify the sample and use it as a model for your own application. Below are some ideas to get you started.

- Update the Cloud Scheduler configuration to specify your own cron messages. You can update the cron job directly, as described in Creating and configuring cron jobs (/scheduler/docs/creating).
- Update `test_executor.py` to run a real script instead of `logger_sample_task.py`, or write your own `Executor` utility.
- Instead of manually launching the utility on Compute Engine and running it as a foreground process, you can launch it automatically as a daemon by a system or third-party tool, such as `systemd` or Supervisor (http://supervisord.org/introduction.html).
- Neither Cloud Scheduler nor Pub/Sub make strict "exactly once" delivery guarantees. While unlikely, duplicate message delivery can occur. If running a specific task more than once creates an undesirable outcome, use a distributed consistent locking tool like Zookeeper (https://zookeeper.apache.org/) to ensure the task is run only once and by only a single instance.
- When scheduling tasks, follow cron best practices and ensure that tasks are scheduled far enough apart they can complete processing before the next time they run.
- Consider whether running Compute Engine instances is necessary for all tasks. An alternative is to trigger Cloud Functions (/functions) in response to Pub/Sub messages. Cloud Functions can call Cloud APIs (/apis), but they cannot execute shell scripts directly. If you need to execute shell scripts, you can call the Compute Engine API to create temporary Compute Engine instances to run scripts. These instances can simply shut themselves down when the tasks finish. This gives you flexibility to complete tasks soon after the event time, at minimal cost.

Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).