

[Private Catalog](https://cloud.google.com/private-catalog/) (https://cloud.google.com/private-catalog/)

[Documentation](https://cloud.google.com/private-catalog/docs/) (https://cloud.google.com/private-catalog/docs/) [Guides](#)

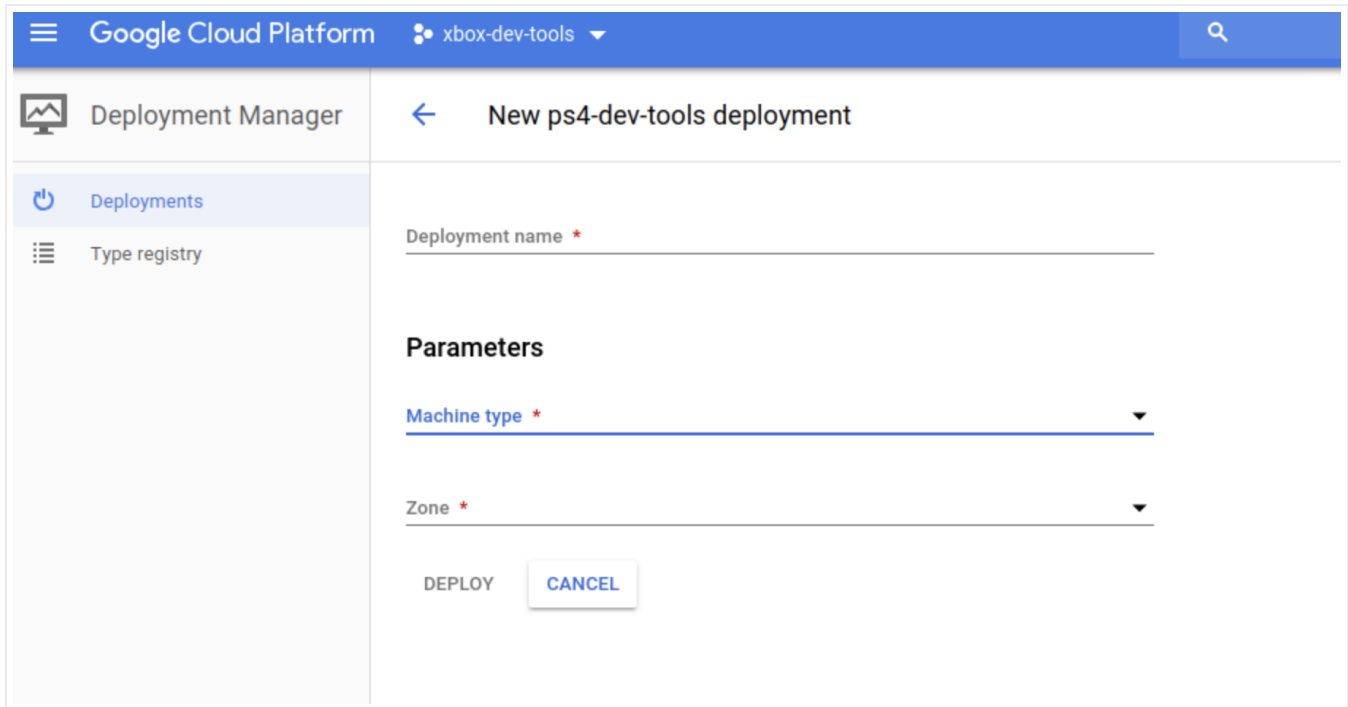
Overview of Form Schema

In Private Catalog, one type of supported solution is Cloud Deployment Manager configurations. To help you create and deploy Cloud Deployment Manager configurations, we have Form Schema.

Form Schema is used for laying out user interface (UI) components in an HTML form. Specifically, it allows cloud admins and developers to provide a UI for entering parameters when creating a new instance or cloud deployment.

Cloud admins use Form Schema to create forms that allow end users to customize Cloud Deployment Manager template-based solutions before launching the solutions. For example, users can select the machine type, disk size, zone, and number of CPUs that a virtual machine will have. These forms look like the forms used in [Google Cloud Marketplace](https://cloud.google.com/marketplace/docs/) (https://cloud.google.com/marketplace/docs/).

The following screenshot illustrates what a deployment form looks like:



The screenshot shows the Google Cloud Platform interface for creating a new deployment. The top navigation bar includes the Google Cloud Platform logo, the account name 'xbox-dev-tools', and a search icon. The left sidebar shows the 'Deployment Manager' section with 'Deployments' and 'Type registry' options. The main content area is titled 'New ps4-dev-tools deployment' and contains the following fields:

- Deployment name ***: A text input field.
- Parameters**: A section header.
- Machine type ***: A dropdown menu.
- Zone ***: A dropdown menu.

At the bottom of the form, there are two buttons: 'DEPLOY' and 'CANCEL'.

Alternatives to Form Schema

Form Schema is an alternative to an existing way of creating UI forms, called Display Metadata.

Compared to Display Metadata, Form Schema has improved flexibility and is open source.

Relationship to JSON Schema

Form Schema builds on [JSON schema form](https://json-schema.org/documentation.html) (https://json-schema.org/documentation.html), which is an open source schema written in JSON for specifying and validating a set of parameters.

Form Schema references fields in the JSON Schema and inherits attributes from it.

You can include Form Schema inside the JSON Schema by including an array of **form entry** objects inside the **form** attribute, as illustrated by the following example:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {...}
```

```
"form": [  
  ...Form List goes here...  
]  
}
```

How to specify a Form Schema instance

You can specify Form Schema instances in YAML format.

You create a schema file that lays out the UI widgets in the order they should appear in the UI. This schema file has the file extension `.py` or `.jinja`, as described in the [Cloud Deployment Manager documentation](#)

(<https://cloud.google.com/deployment-manager/docs/step-by-step-guide/create-a-template>).

You then add the files to a Cloud Deployment Manager template zip archive and upload the zip archive.

How Form Schema fits into the Private Catalog workflow

Cloud admins make use of Form Schema with Private Catalog as follows:

1. Create a Cloud Deployment Manager template.
2. Specify a JSON Schema to define or validate which input fields can be used in the UI for the Cloud Deployment Manager template configuration.
3. Define in Form Schema which fields to include for a particular solution and the order in which those fields should appear. In addition to ordering, you can use Form Schema to provide user-friendly text for enumerated fields, such as dropdowns, and group fields together with section titles.

Form list

An array of `form_entry` (`#form-entry`) objects. Each object represents a UI component in a form. You should specify fields in the order in which you want them to appear in the form, regardless of their location in the schema.

Form entry

A form entry specifies the appearance of a UI component in a form. It can be a string, where it specifies the key for a field in the JSON Schema, or an object. When a form entry is a string, default values for the appearance are inherited from the JSON Schema entry.

When a form entry is an object, the `key` attribute refers to the JSON Schema entry. Use a dot `.` to separate nodes for a nested value. For example, use `name.first` to reference a `first` field inside a `name` object. All other fields are optional, and where specified inherit a default value from the JSON Schema.

Fields	
key*	String Specifies the field definition in the JSON schema.
widget	Widget (#widget) Specifies the UI widget to use for this field. Default: based on mapping the field type (#type-to-widget_mapping).
title	String Title for the field. Inherits title from schema.
notitle	Boolean Whether to hide the title. Default: false.
description	String Used as a hint or tooltip for the field. Inherits description from schema.
validationMessage	String Message to show when the field is invalid.
placeholder	String Placeholder for the field. Note: Material Design uses title as a placeholder instead.
readonly	Boolean Whether field is read only. Inherits readonly from schema.
condition	String A logical expression that determines whether the field is shown.
titleMap	Title map (#title-map) Provides text labels for the options in checkboxes , radio , and select

widgets.

* required

Special handling for types

Object

For type `object`, the field `additionalProperties` defines whether or not extra properties can be present. The field can have a value of `true` (allow anything), `false` (no extra properties allowed), or some JSON schema that constrains the additional properties allowed. When the value is `false` or not present, widgets are displayed for the fields listed in the `items` attribute. For other values (`true` or JSON schema), a textarea is displayed for entering a JSON value.

Type-to-widget mapping

If no widget is specified in the Form Schema, a default value is used based on the JSON Schema Type of the field, as follows:

Schema Type	Schema Form Widget
string	<u>text</u> (#text)
number	<u>number</u> (#number)
integer	<u>number</u> (#number)
boolean	<u>checkbox</u> (#checkbox)
object	fieldset **
string + enum	<u>radio</u> (#radio) (3 or less choices)
string + enum	<u>select</u> (#select) (4 or more choices)
array + enum	<u>checkboxes</u> (#checkbox)
array	<u>array</u> (#array)

** Not supported during beta release

Widget

A widget is specified as a string that refers to one of the following data structures.

Array

A list where rows can be added, removed, and reordered. JSON Schema allows the `items` property for the `array` type to be either a schema or a list of schemas. The list is not supported by Form Schema.

Arrays with primitive items

Because the form schema requires all form entries to have a key, and arrays that contain primitive (non-object) items don't have a key to reference them by, the form should reference the primitive array entry using a reserved keyword: 'x-googleProperty'.

For example, you would define a JSON schema array containing strings as follows:

```
{
  'exampleArray': {
    'type': 'array',
    'items': {
      'type': 'string'
    }
  }
}
```



Should then be referenced in the form schema as:

```
[
  {
    'key': 'exampleArray',
    'widget': 'array',
    'items': [
      {
        'key': 'exampleArray.x-arrayPrimitive'
      }
    ]
  }
]
```



```
    ]  
  },  
]
```

Arrays with object items

Arrays that contain objects should specify each key in the item type that should be rendered in the form.

For example, consider a JSON schema array containing an object:

```
{  
  'exampleArray': {  
    'type': 'array',  
    'items': {  
      'type': 'object',  
      'properties': {  
        'someArrayItemAttribute': {  
          'type': 'string'  
        }  
      }  
    }  
  }  
}
```

The form schema required to create a form array with an input for the `someArrayItemAttribute` attribute would be:

```
[  
  {  
    'key': 'exampleArray',  
    'widget': 'array',  
    'items': [  
      {  
        'key': 'exampleArray.someArrayItemAttribute'  
      }  
    ]  
  },  
]
```

Checkbox

An input field with type `checkbox`.

Checkboxes

A list of input fields of type `checkbox`. The JSON Schema field should be of type `array` and have an `enum` attribute. To provide labels for the options, you can specify a `titleMap` (`#title-map`).

Expander

Very similar to a section, but places fields inside an expansion panel widget that users can open and close by clicking on the title.

Number

An input field with type `number`. The following attributes from the JSON Schema are validators for the field: `minimum`, `maximum`, `exclusiveMinimum`, `exclusiveMaximum`, `multipleOf`.

Password

An input field with type `password`.

Radio

An input field with type `radio`. Use this for fields that have an `enum` list in the JSON Schema, or are of type `boolean`. To provide labels for the options, you can specify a `titleMap` (`#title-map`).

Section

This widget groups together a set of fields. The `key` field is ignored. A section has a required `items` attribute, which is an array of `form entry` (`#form-entry`) objects. A section can have the following optional attributes: `title`, `description`, `condition`.

Select

A `select` input field. Use this for fields that have an `enum` list in the JSON Schema, or are of type `boolean`. To provide labels for the options, you can specify a `titleMap` (`#title-map`).

Text

An input widget with type `text`. If the JSON Schema contains a `pattern` attribute, the pattern is used as a regex validator.

Textarea

A textarea input widget. This widget is displayed in some situations for entering JSON directly. See [object \(#object\)](#).

Title map

The `titleMap` attribute can be specified for widgets of type `checkboxes`, `radio`, and `select`. It's ignored for other widgets.

The attribute is an array of objects that contain two attributes, `value` and `name`. The `value` attribute is a reference to one enum value for the field. The `name` attribute is text to use as a label for the corresponding option in the UI widget. When the widget is `radio` or `checkboxes`, an optional `description` field is added as subtext for that radio button or checkbox.

If no `titleMap` is provided, the enum values are used instead.

Unsupported features

The following features are not currently supported in Form Schema:

- Global options
- Complex validation messages (only one message is supported)
- Validation message interpolation
- Validation message functions
- Custom validation
- Widgets not supported: `actions`, `fieldset`, `radios-inline`, `radiobuttons`, `help`, `template`, `tab`, `tabarray`

- Options not supported: `onChange`, `feedback`, `disabledSuccessState`, `disabledErrorState`, `ngModelOptions`, `htmlClass`, `fieldHtmlClass`, `labelHtmlClass`, `copyValueTo`, `destroyStrategy`
- Post-process function
- Events
- Manual field insertion

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (https://www.apache.org/licenses/LICENSE-2.0). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (https://developers.google.com/terms/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated December 4, 2019.