GCP Certification Series: 3.7 Deploying an Application using Deployment Manager

Prashanta Paudel
Oct 27, 2018 · 6 min read

Deployment manager is a central place to the deployment of machines and instances either using the marketplace or using any other instances from the template using a .yaml file.

Deployment Manager enables you to manage your infrastructure using declarative configurations.

In this codelab, you will use the Deployment Manager to configure network firewall rules and launch a Google Compute Engine instance. You will configure a startup-script that will launch a web server and generate a web page containing instance metadata.

Putting infrastructure into a template and using them repeatedly comes from the concept of "infrastructure as a code".

It is even better to have a code and infrastructure together so anyone can build it again.

Some of the tools for using infrastructure as a code are

1. Ansible
2. Terraform

But the definition on the template may not match with what is available in cloud providers platform.

As each cloud vendors products are changing rapidly they provide their own tool to define cloud infrastructure using declarative language which can be stored outside.

Google Cloud Deployment manager can automate the configuration of all your GCP resources.

Deployment Manager Basics(ref: https://medium.com/google-cloud/2018-google-deployment-manager-5ebb8759a122)

Deployment Manager has the concept of a *deployment*, which is a collection of GCP resources that form a logical unit and that are deployed together. The resources of a deployment can be anything available on GCP: VMs, IP addresses, database servers, Kubernetes clusters, etc.

In order to create a deployment, you need a *deployment configuration*, which is a YAML file containing the definition of the resources. To give you an idea, it could look as follows:

```
resources:
- name: example-vm
  type: compute.v1.instance
  properties:
    zone: europe-west1-b
    machineType: zones/europe-west1-b/machineTypes/n1-
standard-1
    disks:
       ...
```

Each listed resource always has a *type*, which is the kind of resource that will be created (a VM, an IP address, etc.), a *name*, and *properties* describing with what parameters the resource should be created.

An important concept, in comparison to say Ansible, is that when you create a deployment, it exists as a resource itself inside GCP. If you later change the configuration of the deployment (by modifying the YAML file, for example), and run the update command, deployment manager will compare the new configuration to what it deployed before, and will only make the required changes. Not only less work needs to be done, but more importantly, it also ensures that any removed resources from the configuration, will also be removed from the GCP infrastructure.

You can find many deployment configuration examples at the Google Deployment Manager samples project on GitHub. It is often more useful than the official documentation, to find out exactly what parameters you should define.

The cluster.yaml file in the GitHub repository specifies a few more parameters, such as distributing the nodes across two zones and enabling “auto-upgrade” for the cluster nodes.

You can instantiate this deployment by using the “create” command:

```
$ gcloud deployment-manager deployments create example-
cluster --config cluster-1/cluster.yaml
```

This creates a deployment called “example-cluster”, using the definition found in cluster-1/cluster.yaml. You can later update the yaml file, and update the deployed deployment with the “update” command:

```
$ gcloud deployment-manager deployments update example-
cluster --config cluster-1/cluster.yaml
```

This command compares the deployed deployment with the new definition and executes only the needed changes (for example creates a new resource that you added).

For now, delete the cluster. We will re-create it again later.

```
$ gcloud deployment-manager deployments delete example-
cluster
```

Infrastructure Template

Yaml files created in standard format actually code for your infrastructure.

Deployment Manager supports

- Python scripts
- Jinja 2 as templating language.

Deployment Manager Fundamentals

Configuration

A configuration describes all the resources you want for a single deployment. A configuration is a file written in YAML syntax that lists each of the resources you want to create and its respective resource properties. A configuration must contain a `resources:` section followed by the list of resources to create.

Each resource must contain three components:

- `name` - A user-defined string to identify this resource such as `my-vm`, `project-data-disk`, `the-test-network`.
- `type` - The type of resource being deployed such as `compute.v1.instance`, `compute.v1.disk`. The base resource types are described and listed on the Supported Resource Types documentation.
- `properties` - The parameters for this resource type. They must match the properties for the type such as `zone: asia-east1-a`, `boot: true`.

```
resources:
- name: the-first-vm
  type: compute.v1.instance
  properties:
    zone: us-central1-a
    machineType:
https://www.googleapis.com/compute/v1/projects/myproject/zon
es/us-central1-a/machineTypes/f1-micro
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage:
https://www.googleapis.com/compute/v1/projects/debian-
cloud/global/images/debian-7-wheezy-v20150423
```

```
    networkInterfaces:
    - network:
https://www.googleapis.com/compute/v1/projects/myproject/glo
bal/networks/default
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT
```

Templates

A configuration can contain templates, which are essential parts of the configuration file that has been abstracted into individual building blocks. After you create a template, you can reuse them across deployments as necessary. Similarly, if you find yourself rewriting configurations that share very similar properties, you can abstract the shared parts into templates. Templates are much more flexible than individual configuration files and intended to support easy portability across deployments.

A template is a file is written in either Python or Jinja2. The Deployment Manager system will interpret each template recursively and inline the results with the configuration file. As such, the interpretation of each template eventually must result in the same YAML syntax for resources as that defined above for the configuration file itself.

To create a simple template, read Creating a Basic Template.

Configurations are described as fully-expanded or unexpanded. A fully-expanded configuration describes all resources and properties of the deployment, including any content from imported template files. For example, you would supply an unexpanded configuration that uses a template like so:

```
imports:
- path: vm_template.jinja

resources:
- name: vm-instance
  type: vm_template.jinja
  properties:
    zone: us-central1-a
    project: myproject
```

Once expanded, your configuration file would contain the contents of all your templates, like so:

```
resources:
- name: the-first-vm
  type: compute.v1.instance
  properties:
    zone: us-central1-a
    machineType:
https://www.googleapis.com/compute/v1/projects/myproject/zon
es/us-central1-a/machineTypes/f1-micro
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage:
https://www.googleapis.com/compute/v1/projects/debian-
cloud/global/images/debian-7-wheezy-v20150423
        networkInterfaces:
        - network:
https://www.googleapis.com/compute/v1/projects/myproject/glo
bal/networks/default
    accessConfigs:
    - name: External NAT
      type: ONE_TO_ONE_NAT
```

Deployment

A deployment is a collection of resources that are deployed and managed together, using a configuration.

———————————————————————————————————— -

Creating a Basic Template

A basic configuration file might be enough for simple workloads but for more complex architectures or for configurations that you plan to reuse, you can break your configuration into templates.

A template is a separate file that is imported and used as a type in a configuration. You can use as many templates as you want in a configuration and Deployment Manager will recursively expand any imported templates to create your full configuration.

Templates allow you to separate your configuration out into different pieces that you can use and reuse across different deployments. Templates can be as generalized or specific as you need. With templates, you can also take advantage of features like template properties, environment variables, modules, and other template functionality to create dynamic configuration and template files.

Creating a Basic Configuration

A configuration file defines all the Google Cloud Platform resources that make up a deployment. You must have a configuration file to create a deployment. A configuration file must be written in YAML syntax.

Configuration file structure

A configuration file is written in YAML format and has the following structure:

```
imports:
- path: path/to/template.jinja
  name: my-template
- path: path/to/another/template.py


resources:
  - name: NAME_OF_RESOURCE
    type: TYPE_OF_RESOURCE
    properties:
      property-a: value
      property-b: value
      ...
      property-z: value
  - name: NAME_OF_RESOURCE
    type: TYPE_OF_RESOURCE
    properties:
      property-a: value
      property-b: value
      ...
      property-z: value
```

Each of the sections define a different part of the deployment:

- The `imports` sections is a list of template files that will be used by the configuration. Deployment Manager recursively expands any imported templates to form your final configuration.
- The `resources` section is a list of resources that make up this deployment. A resource can be:
- A Google-managed base type, such a Compute Engine VM instance.
- An imported template
- A composite type.
- A type provider.

You can also include other optional sections, such as the `outputs` and `metadata` sections. The `outputs` section allows you to expose data from your templates and configurations as outputs for other templates in the same deployment to consume or as outputs for your end users, while the `metadata` section lets you use other features, likesetting explicit dependencies between resources.

At the minimum, a configuration must always declare the `resources` section, followed by a list of resources. Other sections are optional.

resources:

— name : [name of resource]

type: compute.v1.instance

Properties:

————————

———————— —

———————-