Google Cloud Platform for AWS Professionals: Infrastructure Deployment Tools

Updated December 6, 2017

On both Google Cloud Platform (GCP) and Amazon Web Services (AWS), you can manage your cloud environments using an infrastructure-as-code approach. GCP offers Deployment Manager (https://cloud.google.com/deployment-manager/), and AWS offers CloudFormation (https://aws.amazon.com/cloudformation/).

Treating your infrastructure as code involves defining your environment through scripts, templates, and configuration files, and keeping the files in a source code repository. Using this approach, you can:

- Control the version of your configuration.
- Deploy consistent, reproducible configurations.
- Review an audit trail of changes to your configuration.
- Use the configuration as part of a continuous deployment system.
- Easily fail back to a previous known-good configuration.

Service model comparison

The following table maps high-level CloudFormation features, terms, and concepts to Deployment Manager features, terms, and concepts:

Feature	AWS CloudFormation	GCP Deployment Manager
Deployed collection of resources	Stack	Deployment
Deployment files	Template file	Configuration files, template files, and schema files
Syntax	JSON, YAML	YAML, Jinja, Python
Composition and reuse	Nested stacks	Templates
Identification of individual resources	Logical ID	Name

Feature	AWS CloudFormation	GCP Deployment Manager
Scope of deployment locality	Regional	Global
Default maximum number of stacks or deployments	200	1000
Graphical user interface	Yes	No
Preview	Yes	Yes
Stack policies	Yes	No

Templates and configuration

The following table compares available template and configuration features in CloudFormation to those in Deployment Manager:

Feature	AWS CloudFormation	GCP Deployment Manager
Maximum size of template or configuration	460.8 KB	1 MB
Declarative	Yes	Yes
Conditionals	Yes	Yes
Loops	No	Yes
Parameterization	Yes	Yes
Output values	Yes	Yes

AWS CloudFormation

In AWS CloudFormation, you create a template (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html) that defines a set of actions against various services, such as creating an S3 bucket or launching an EC2 instance. An AWS template can be expressed in YAML or JSON, and AWS CloudFormation can invoke a template from either an S3 bucket or your local machine. When

AWS CloudFormation invokes a template, it makes calls to the APIs of the services you have defined in the template to launch and configure the services.

To use the services defined in a template, you must have appropriate AWS IAM permissions for each service. You can also set IAM policies (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-iam-template.html) to restrict or allow the use of CloudFormation.

Deployment Manager

Deployment Manager uses three types of files to define a deployment:

- A configuration file (https://cloud.google.com/deployment-manager/docs/configuration/), which defines the structure of your deployment and configures the resources to be deployed. This file is written in YAML. You can use this file as a standalone configuration for your deployment, or you can combine it with imported template files for increased flexibility.
- Template files (https://cloud.google.com/deployment-manager/docs/configuration/templates/create-basic-template), which are composable, reusable resource definitions. Template files are written in Jinja 2.7.3 (http://jinja.pocoo.org/docs/dev/) or Python 2.7 (https://www.python.org/download/releases/2.7/). You can add loops to create multiple instances of the same resource without having to explicitly declare each instance.
- Schema files (https://cloud.google.com/deployment-manager/configuration/using-schemas), which define sets of rules that a configuration file must meet in order to use a particular template.

You pass the configuration file (which will expand any templates that it references) to the Deployment Manager service through the `gcloud` command-line tool or API. You can also launch predefined configurations by using GCP Marketplace (https://cloud.google.com/deployment-manager/docs/deployment-manager-and-cloud-marketplace).

By default, Deployment Manager uses the credentials of the GCP API service account (https://cloud.google.com/compute/docs/access/service-accounts) to authenticate to other APIs. You must grant the user using Deployment Manager the appropriate permissions to use Deployment Manager. You can use predefined IAM roles (https://cloud.google.com/deployment-manager/docs/access-control#predefined_roles) to limit the permissions appropriately. Deployment Manager allows you to automate the creation of

projects
(https://github.com/GoogleCloudPlatform/deploymentmanager-samples/tree/master/examples/v2/project_creation)
themselves as well as the resources contained within the projects.

A Deployment Manager configuration file consists of the following sections:

```
imports:
  - path: network.jinja
resources:
- name: network
  type: network.jinja
  properties:
    region: us-central1
    subnets:
      - range: 10.177.0.0/17
        name: web
      - range: 10.177.128.0/17
        name: data
- name: web-instance
  type: compute.v1.instance
  properties:
    zone: us-central1-f
    machineType: https://www.googleapis.com/compute/v1/projects/myproject/zones/
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: https://www.googleapis.com/compute/v1/projects/debian-cloud
    networkInterfaces:
    - network: https://www.googleapis.com/compute/v1/projects/myproject/global/n
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT
```

The above GCP example configuration consists of the following parts:

Imports. This references the path to composable templates that are expanded as part of the actual deployment. This is optional, because you might have a simple deployment where you do not need to create a configuration where you have placed individual resources into composite template files

Resources. This is a list of the resources that you want deployed as part of your configuration. This is the only mandatory requirement for a configuration. Each resource is defined by the following constituent parts:

- **Name**. A name that you define for this resource. In the example above we have two resources called `network` and `web-instance`.
- **Type**. Specifies the base type of the resource. The example shows two types of resource. The `web-instance` is of type `compute.v1.instance`. The second resource refers to the import file declared in the `imports` section of the configuration file. In the example it is a Jinja file type, but you can also import Python files.
- **Properties**. These are the properties associated with a resource. The resource called `web-instance` defines a Compute Engine instance of machine type `f1-micro,` which has a persistent disk that can be auto deleted when you delete the instance. The instance is to be deployed into `us-central1-f` region. The second resource is defined in a separate template file called `network.jinja`. For this resource, you define the values for the properties in the configuration file. The values are passed through to the actual template file. This composability lends itself to building complex deployments with reusable resource definitions, to which you can pass values through the configuration files. For example, you could use the same template file to create more subnets with different IP ranges and names.

A configuration file can also have **outputs** (https://cloud.google.com/deployment-manager/docs/configuration/syntax-reference#outputs) and **metadata** (https://cloud.google.com/deployment-manager/docs/configuration/syntax-reference#metadata).

A **schema file** defines a set of rules for how to use a template. Defining the rules in a single schema gives your users a way to interact with the templates you write. They can use the schema file without needing to examine the templates that constitute a configuration or to learn which properties need to be set and which have default values. For more information on schema files, see the Deployment Manager documentation (https://cloud.google.com/deployment-manager/docs/configuration/using-schemas).

The example template uses a public image to create the instance. GCP takes a global approach, so when you select an image type to launch an instance, you do not need to create a map of the image type per region, as you would for AWS CloudFormation. Using GCP, you declare the machine type and the source image. The source image is available to all projects.

Composition and reuse

Both AWS CloudFormation and GCP Deployment Manager encourage you to compose your own stacks or deployments and to reuse stacks or deployments.

AWS CloudFormation

AWS CloudFormation relies on the concept of nested stacks. With nested stacks, you can create a template for the resources that you want to reuse. You store the template that you want to call from the master template in an S3 bucket and use the `AWS::CloudFormation::Stack` type. You pass the S3 URL of the template as a property.

You can use parameters to easily customise a template each time you use it to create a stack. You can use intrinsic functions (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html) together with parameters to create conditional resources.

You can use outputs (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html) to export a value through the logical ID assigned to the value, so that you can use the value elsewhere. For example, you can export the names of load balancers and S3 buckets, import values into other stacks, and expose the values in the console.

CloudFormation creates resources in parallel and determines which resources can be deployed in parallel, but you might need to use the creation policy and `DependsOn` attributes to help enforce the order in which resources are created. For example, the deployment might require EBS volumes to be created before the instance is created that uses the volumes.

You can use custom resources to include non-AWS resources as part of your CloudFormation stack. To implement a custom resource, you supply a service token in the `Properties` section of the template. The service token specifies where CloudFormation should send requests.

GCP Deployment Manager

Deployment Manager allows you to import templates into your configuration file. You declare the path to the templates, which can be stored locally, or in a Cloud Storage bucket, or at a publicly accessible URL (https://cloud.google.com/deployment-manager/docs/configuration/hosting-templates-externally) such as GitHub, in the `import` statement. This allows you to scale and reuse templates as part of your configuration.

You can customize Deployment Manager templates by passing variables through to your templates. You can use two types of variables with Deployment Manager: template variables (https://cloud.google.com/deployment-manager/docs/configuration/adding-templates#template_variables)
and environment variables
(https://cloud.google.com/deployment-manager/docs/configuration/adding-templates#environment_variables)
. Using template variables, you define the customizable value in the configuration file and pass that through to the template files. Environment variables are predefined and automatically assigned. Environment variables are designed to be used for items that are not directly related to resources you want to deploy, such as project IDs and deployment names.

Both Jinja and Python allow you to take advantage of their native conditionals and looping functions.

Deployment Manager relies on the concept of template modules
(https://cloud.google.com/deployment-manager/docs/configuration/adding-templates#template_modules)
. You can use template modules to incorporate template snippets that can be used in multiple templates. For example, you can use a template module to always generate instance names based on their environment and function.

Deployment Manager creates all resources in parallel. You can use references
(https://cloud.google.com/deployment-manager/docs/step-by-step-guide/using-references) to enforce the order in which resources are created. For example, if an instance references a new network in a configuration, you can ensure that Deployment Manager always creates the network before creating the instance, no matter where in the configuration file the network is declared.

Outputs (https://cloud.google.com/deployment-manager/docs/configuration/outputs) in Deployment Manager are designed to expose key properties of resources in an easily consumable way, so that they can be used by other templates or be exposed in the configuration. Outputs are declared as key-value pairs and can consist of a static string, a reference
(https://cloud.google.com/deployment-manager/docs/configuration/create-configuration-file#referencing_resource_properties)
to a property, a template variable
(https://cloud.google.com/deployment-manager/docs/configuration/adding-templates#template_variables)
, or an environment variable

(https://cloud.google.com/deployment-manager/docs/configuration/adding-templates#environment_variables)
.

If the GCP-supported types don't meet your needs, you can create a type provider (https://cloud.google.com/deployment-manager/docs/fundamentals#basetypes) and create additional types that you can register with Deployment Manager. To create a type provider, you must supply an API descriptor document, which can be an OpenAPI specification (https://github.com/OAI/OpenAPI-Specification) or a Google Discovery (https://developers.google.com/discovery/v1/reference/apis) document. When you create and register a type provider, all resources provided by the API and supported by a RESTful interface with create, read, update, and delete (CRUD) (https://cloud.google.com/deployment-manager/docs/configuration/type-providers/api-requirements) operations are exposed as types that you can use in your deployment.

You can extend the base types to create a composite type (https://cloud.google.com/deployment-manager/docs/configuration/composite-types/creating-composite-types)
. A composite type is a combination of multiple base types (https://cloud.google.com/deployment-manager/docs/fundamentals#base_types) that are configured to work together. For example, a network load-balanced managed instance group is a composite type. A network load balancer requires multiple GCP resources and some configuration between resources. You can set up these resources in a configuration once and register the type with Deployment Manager.

Metadata, helper scripts, and startup scripts

AWS CloudFormation allows you to use a set of Python scripts that help you start services and install software on EC2 instances. The scripts are available from a repository or installed on an Amazon Linux AMI, and you can call them from your templates. You can add metadata or retrieve metadata on your instances by using the `metadata` attribute.

GCP Deployment Manager allows you to use startup scripts (https://cloud.google.com/deployment-manager/docs/step-by-step-guide/setting-metadata-and-startup-scripts)
or add metadata to virtual machine instances in your deployment by specifying the metadata in your template or configuration. The metadata key for your startup script must be `startup-script`, and the value is the contents of your startup script.

The example snippet below shows how this is done:

```
resources:
- name: my-first-vm-template
  type: compute.v1.instance
  properties:
   zone: us-central1-f
   machineType:
   ...[snip]...
   metadata:
     items:
     - key: startup-script
       value: "STARTUP-SCRIPT-CONTENTS"
```

Updating stacks and deployments

AWS CloudFormation

You can update CloudFormation stacks by passing new parameter values or updated templates. You can use change sets to see which stack settings and resources will change before you apply the change set.

Cloud Deployment Manager

With Deployment Manager you can also update deployments and preview potential changes (https://cloud.google.com/deployment-manager/docs/deployments/updating-deployments#optional_preview_an_updated_configuration) to assess the effect your changes will have. Deployment Manager does not instantiate any actual resources when you preview a configuration. Instead, it expands the full configuration and creates shell resources. This gives you the opportunity to see the deployment before committing to it.

Deployment Manager also provides policies (https://cloud.google.com/deployment-manager/docs/deployments/updating-deployments#optional_determine_the_update_policies_to_use) that you can use when you update a deployment to help you control how Deployment Manager manages updates. Some resources are immutable and might not have an update method. As with AWS CloudFormation, you should check the API methods for the resource to understand the implications of carrying out an update.

You can also look at manifests (https://cloud.google.com/deployment-manager/docs/deployments/viewing-manifest) prior to doing any updates to understand what a deployment actually looks like. You can do this through the command-line interface or the console.

Costs

There is no charge for AWS CloudFormation or GCP Deployment Manager. In both cases you are charged only for the resources you launch.

What's next?

Check out the other Google Cloud Platform for AWS Professionals articles:

- Overview (https://cloud.google.com/docs/compare/aws)
- Compute (https://cloud.google.com/docs/compare/aws/compute)
- Networking (https://cloud.google.com/docs/compare/aws/networking)
- Storage (https://cloud.google.com/docs/compare/aws/storage)
- Big Data (https://cloud.google.com/docs/compare/aws/big-data)
- Management (https://cloud.google.com/docs/compare/aws/management)
- Mobile (https://cloud.google.com/docs/compare/aws/mobile)
- Application Services (https://cloud.google.com/docs/compare/aws/application-services)