# Exposing Information Using Outputs

When you create a deployment, you might want to expose key properties of your configurations or templates for other templates or users to consume. For example, you might want to expose the IP address of a database created in a template so users can easily reference the IP when configuring their own templates.

You can use the **outputs** section in your template or configuration to define a list of key/values pairs that users can call. In the outputs section, you define arbitrary keys and set the value of the keys to a reference (/deployment-manager/docs/configuration/use-references), a template property (/deployment-manager/docs/configuration/templates/define-template-properties), or an environment variable (/deployment-manager/docs/configuration/templates/use-environment-variables). Users can consume outputs to access key information about resources created by the template. For example, you can declare an output called `databaseIP` that references the IP address of an instance hosting a database and users can reference that output in other templates in the same deployment.

## Before you begin

- If you want to use the command-line examples in this guide, install the `gcloud` command-line tool (/sdk).
- If you want to use the API examples in this guide, set up API access (/deployment-manager/docs/reference/latest).
- Understand how to create a basic configuration (/deployment-manager/docs/configuration/create-basic-configuration).

## Example

Here is an example template with outputs:

db.jinja

```
t MASTER = env["name"] + "-" + env["deployment"] + "-mongodb" %}
rces:
```

```
e: {{ MASTER }}  type: instance

ts:
e: databaseIp
ue: $(ref.{{ MASTER }}.network[0].ip)  # Treated as a string during expansion
e: databasePort
ue: 88
```

The outputs section declares two properties: `databaseIp` and `databasePort`. `databaseIp` uses a reference that resolves to the network IP address of the master resource, while `databasePort` is a static value. In another template, you can import `mongodb.jinja`, use the template as a type, and call the outputs. For example:

```
ts:
h: example/path/to/mongodb.jinja
e: mongodb.jinja

rces:
e: my_mongo
e: mongodb.jinja
perties:
ize: 100

e: my_instance
e: compute.v1.instance
perties:

atabaseIp: $(ref.my_mongo.databaseIp)
atabasePort: $(ref.my_mongo.databasePort)
```

## Declaring an output

Declare an output in either a template or a configuration by defining an `outputs:` section at the same level as the `resources:` section. Output keys must be unique within the template or configuration.

For example, a sample `outputs:` section might look like this:

```
ts:
e: databaseIp
ue: $(ref.my-first-vm.networkInterfaces[0].accessConfigs[0].natIP)
e: machineType
ue: {{ properties['machineType'] }}- name: databasePort
ue: 88
```

Here's how the outputs might look in a full template:

examples/v2/build_configuration/use_outputs/template_with_outputs.jinja (https://github.com/GoogleCloudPlatform/deployment-manager-samples/blob/master/examples/v2/build_configuration/use_outputs/template_with_outputs.jinja)

nanager-samples/blob/master/examples/v2/build_configuration/use_outputs/template_with_outputs.jinja)

```
resources:
- name: my-first-vm
  type: compute.v1.instance
  properties:
    zone: us-central1-a
    machineType: zones/us-central1-a/machineTypes/{{ properties['machineType'] }}
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: projects/debian-cloud/global/images/family/debian-9
    networkInterfaces:
    - network: global/networks/default
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT

# Declare outputs here
outputs:
- name: databaseIp
  value: $(ref.my-first-vm.networkInterfaces[0].accessConfigs[0].natIP)
- name: machineType
  value: {{ properties['machineType'] }}- name: databasePort
```

```
  value: 88
```

Output values can be:

- A static string
- A reference (/deployment-manager/docs/configuration/create-configuration-file#referencing_resource_properties) to a property
- A template property (/deployment-manager/docs/configuration/templates/define-template-properties)
- An environment variable (/deployment-manager/docs/configuration/templates/use-environment-variables)

## Using outputs from templates

To use an output that has been defined in a template, import and use the template containing the output as a type. For example, to use outputs defined in a template called `template_with_outputs.jinja`, it must be imported and used to create a resource:

examples/v2/build_configuration/use_outputs/use_template_with_outputs.yaml (https://github.com/GoogleCloudPlatform/deploymentmanager-samples/blob/master/examples/v2/build_configuration/use_outputs/use_template_with_outputs.yaml)

ger-samples/blob/master/examples/v2/build_configuration/use_outputs/use_template_with_outputs.yaml)

```
# Copyright 2016 Google Inc. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
imports:
- path: template_with_outputs.jinja
  name: template.jinja

resources:
- name: my-first-vm
  type: template.jinja
  properties:
    machineType: n1-standard-1

outputs:
- name: databaseIp
  value: $(ref.my-first-vm.databaseIp)
- name: machineType
  value: $(ref.my-first-vm.machineType)
- name: databasePort
  value: $(ref.my-first-vm.databasePort)
```

To call an output, use the following format:

```
.RESOURCE.OUTPUT)
```

- `RESOURCE` is the name of the resource created by the template. In the example above, this is `my-first-vm`.
- `OUTPUT` is the output declared in the template. In the example above, this would be `databaseIp` and `databasePort`. This is the same syntax that you use to declare references. You can reference list items as well, for example: `$ref.template.property[0]`.

When you deploy the configuration, Deployment Manager expands the configuration, and then replaces references to outputs with the output values.

## Describing outputs in schemas

For templates that have accompanying schemas, you can describe output properties in further details. Deployment Manager does not enforce or validate any information in the outputs section but it is potentially helpful to use this section to provide more information about relevant outputs, for the benefit of users using your templates.

In your schema file, provide an outputs section that matches the output in your template. For example:

```
ts:
abaseIp:
escription: Reference to ip address of your new cluster
ype: string
abasePort:
escription: Port to talk on
ype: integer
```

Users can reference your schema file to understand the usage and type of your outputs.

## Looking up final output values

After you deploy templates that use outputs, view the final output values by viewing the configuration layout (/deployment-manager/docs/deployments/viewing-manifest#configuration_layout) of the deployment. Final output values are indicated by the `finalValue` property. All output values are included in this field, including output values from nested templates. For example:

```
t: |
ources:
ame: vm_template
utputs:
 finalValue: 104.197.69.69
 name: databaseIp
 value: $(ref.vm-test.networkInterfaces[0].accessConfigs[0].natIP)
roperties:
 zone: us-central1-a
```

```
esources:
 name: datadisk-example-instance
 type: compute.v1.disk
 name: vm-test
 type: compute.v1.instance
ype: vm_template.jinja
 manifest-1455057116997
```

## Avoid circular dependencies

Be careful when creating templates where two or more resources rely on outputs from each other. Deployment Manager does not prevent this structure but if the outputs caused a circular dependency, the deployment won't deploy successfully. For example, the following snippet is accepted by Deployment Manager but if the contents of the templates causes a circular dependency, the deployment would fail:

```
rces:
e: frontend
e: frontend.jinja
perties:
p: $(ref.backend.ip)
e: backend
e: backend.jinja
perties:
p: $(ref.frontend.ip)
```

As an example of a circular dependency where the deployment fails, assume both frontend.jinja and backend.jinja looked like this:

```
rces:
e: {{ env['name'] }}  type: compute.v1.instance
perties:
one: us-central1-f
..
etworkInterfaces:
 network: global/networks/default
 accessConfigs:
```

```
 - name: External NAT
   type: ONE_TO_ONE_NAT
etadata:
 items:
 - key: startup-script
   value: |
     #!/bin/bash
     export IP={{ properties["ip"] }}      ...

ts:
e: ip
ue: $(ref.{{ env['name'] }}.networkInterfaces[0]accessConfigs[0].natIP)
```

Recall that both resources used the IP output property from the opposing resource:

```
rces:
e: frontend
e: frontend.jinja
perties:
p: $(ref.backend.ip)
e: backend
e: backend.jinja
perties:
p: $(ref.frontend.ip)
```

But neither IP values can be populated because both properties rely on the existence of the other resource, creating a circular dependency. Here is the same template, fully expanded:

```
rces:
e: frontend
e: compute.v1.instance
perties:
one: us-central1-f
..
etworkInterfaces:
 network: global/networks/default
 accessConfigs:
 - name: External NAT
   type: ONE_TO_ONE_NAT
etadata:
```

```
 items:
 - key: startup-script
   value: |
     #!/bin/bash
     export IP=$(ref.backend.networkInterfaces[0]accessConfigs[0].natIP)
e: backend
e: compute.v1.instance
perties:
one: us-central1-f
..
etworkInterfaces:
 network: global/networks/default
 accessConfigs:
 - name: External NAT
   type: ONE_TO_ONE_NAT
etadata:
 items:
 - key: startup-script
   value: |
     #!/bin/bash
     export IP=$(ref.frontend.networkInterfaces[0]accessConfigs[0].natIP)
```

Deployment Manager returns an error if you try to run configuration:

```
: u'CONDITION_NOT_MET'
age: u'A dependency cycle was found amongst backend, frontend.'>]>
```

However, this template would work if:

1. frontend.jinja created two virtual machine instances, vm-1 and vm-2.
2. backend.jinja created vm-3 and vm-4.
3. vm-1 exposed it's external IP as an output and vm-4 used that output.
4. vm-3 exposed an external IP as an output, vm-2 used that output.

## What's next

- Create a deployment (/deployment-manager/docs/deployments).

- Learn more about templates (/deployment-manager/docs/configuration/templates/create-basic-template).



Last updated 2020-07-27 UTC.