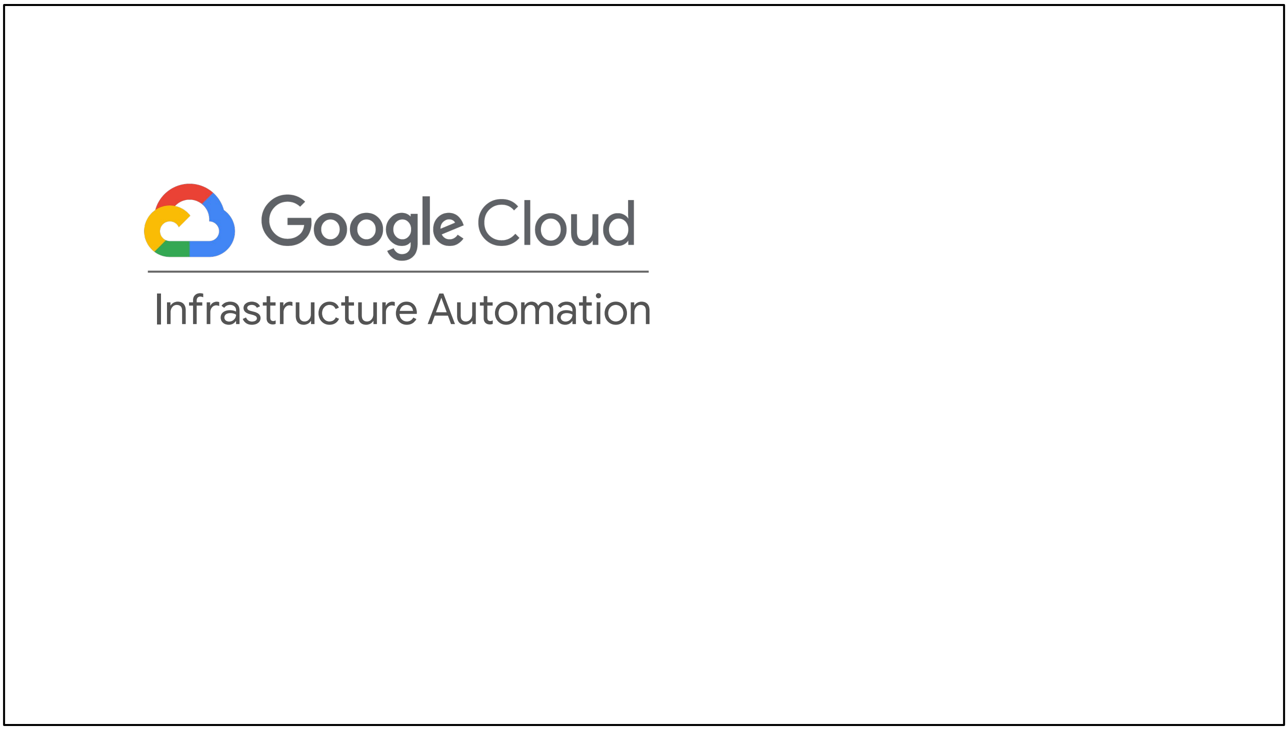


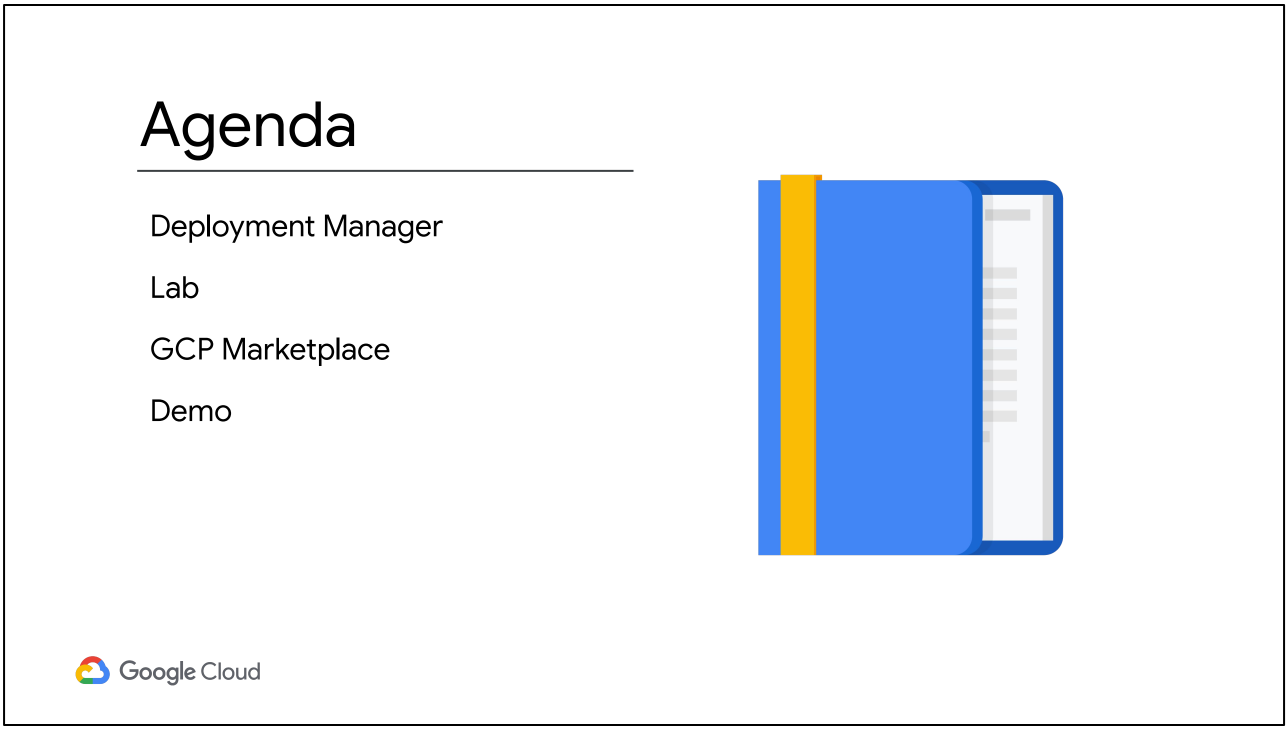
Google Cloud

Infrastructure Automation

Now that we have covered several of GCP’s services and features, it makes sense to talk about how to automate the deployment of GCP infrastructure.

Calling the cloud API from code is a powerful way to generate infrastructure. But writing code to create infrastructure also has some challenges. One issue is that the maintainability of the infrastructure depends directly on the quality of the software. For example, a program could have a dozen locations that call the cloud API to create VMs. Fixing a problem with the definition of one VM would require first identifying which of the dozen calls actually created it. Standard software development best practices will apply, and it’s important to note that things could change rapidly, requiring maintenance on your code.

Clearly another level of organization is needed. That's the purpose of Deployment Manager. Deployment Manager uses a system of highly structured templates and configuration files to document the infrastructure in an easily readable and understandable format. Deployment Manager conceals the actual Cloud API calls, so you don't need to write code and can focus on the definition of the infrastructure.



Agenda

Deployment Manager

Lab

GCP Marketplace

Demo

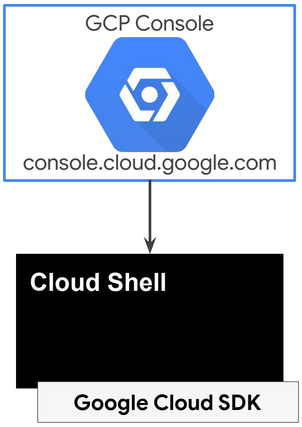
Google Cloud

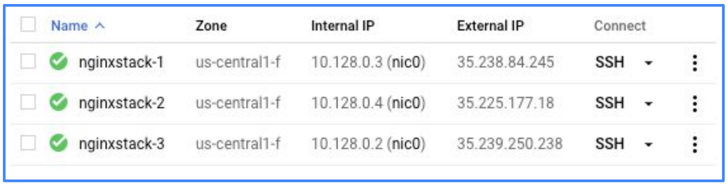
In this module, we cover how to use Deployment Manager to automate the deployment of infrastructure and how to use GCP Marketplace to launch infrastructure solutions.

You will use Deployment Manager or Terraform to deploy a VPC network, a firewall rule, and VM instances in the lab of this module. I will also demonstrate how to launch infrastructure solutions using GCP Marketplace.

Let’s start by talking about Deployment Manager.

GCP Console, Cloud SDK, and Cloud Shell



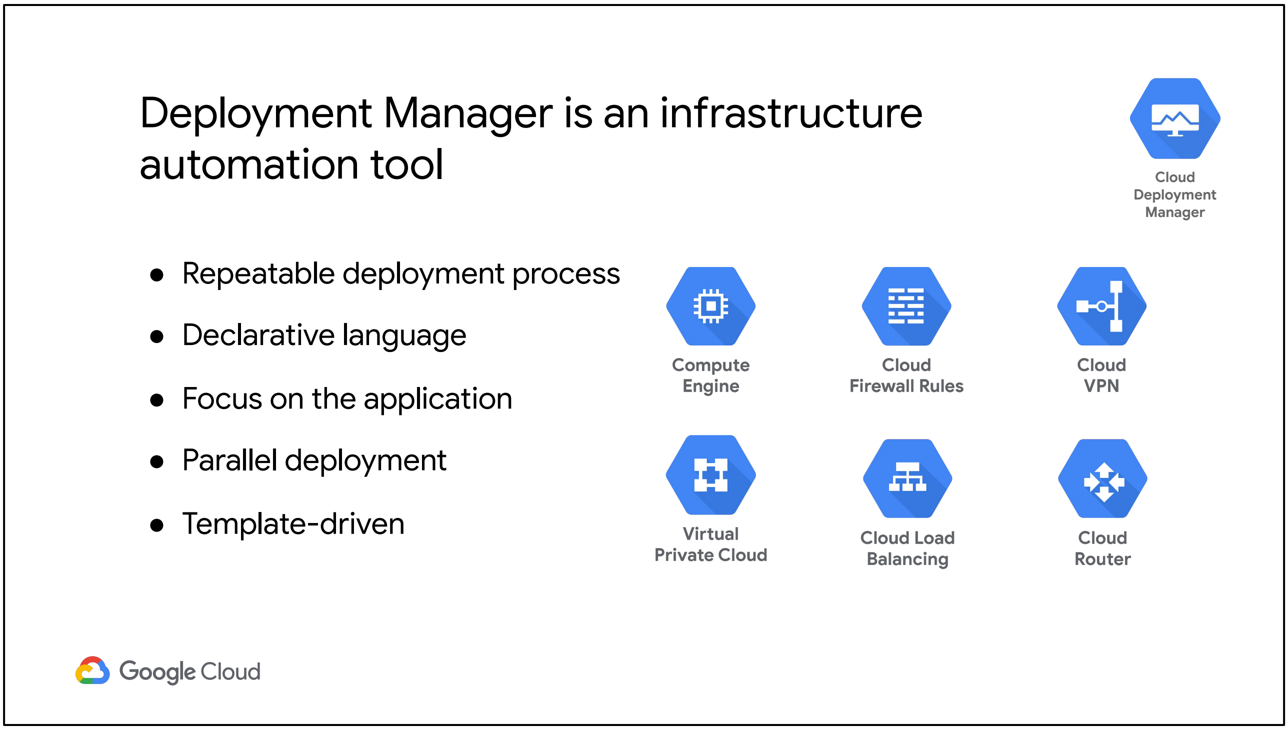


□	Name ^	Zone	Internal IP	External IP	Connect	
□	✓ nginxstack-1	us-central1-f	10.128.0.3 (nic0)	35.238.84.245	SSH ▾	⋮
□	✓ nginxstack-2	us-central1-f	10.128.0.4 (nic0)	35.225.177.18	SSH ▾	⋮
□	✓ nginxstack-3	us-central1-f	10.128.0.2 (nic0)	35.239.250.238	SSH ▾	⋮

```
$ gcloud compute instances list
NAME          ZONE           INTERNAL_IP   EXTERNAL_IP
nginxstack-1  us-central1-f  10.128.0.3    35.238.84.245
nginxstack-2  us-central1-f  10.128.0.4    35.225.177.18
nginxstack-3  us-central1-f  10.128.0.2    35.239.250.238
```

Google Cloud

So far, you have been creating GCP resources using the GCP Console and Cloud Shell. I recommend the GCP Console when you are new to using a service or if you prefer a UI. Cloud Shell works best when you are comfortable using a specific service and you want to quickly create resources using the command line. Deployment Manager takes this one step further.



Deployment Manager is an infrastructure automation tool

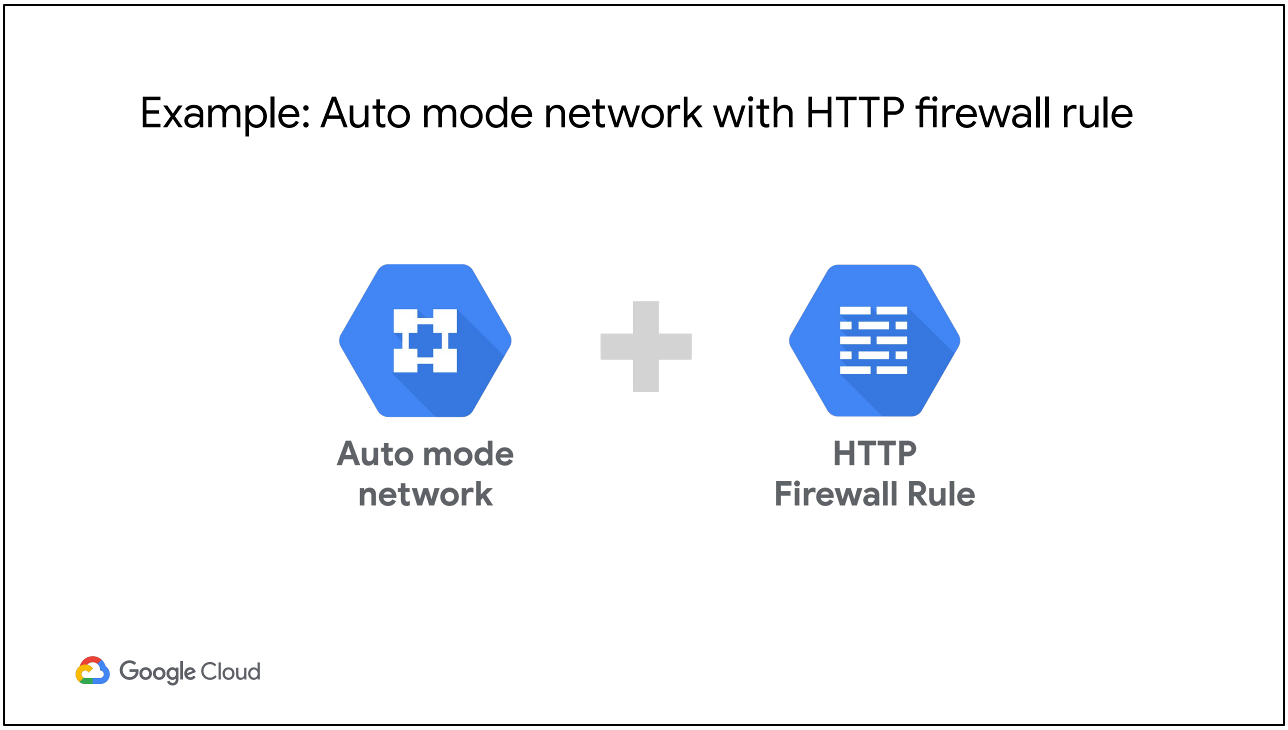
- Repeatable deployment process
- Declarative language
- Focus on the application
- Parallel deployment
- Template-driven

Cloud Deployment Manager
Compute Engine
Cloud Firewall Rules
Cloud VPN
Virtual Private Cloud
Cloud Load Balancing
Cloud Router
Google Cloud

Deployment Manager is an infrastructure deployment service that automates the creation and management of GCP resources for you. You just specify all the resources needed for your application in a declarative format and deploy your configuration. This deployment can be repeated over and over with consistent results, and you can delete a whole deployment with one command or click. The benefit of a declarative approach is that it allows you to specify what the configuration should be and let the system figure out the steps to take.

Instead of deploying each resource separately, you specify the set of resources which compose the application or service, allowing you to focus on the application. Unlike Cloud Shell, Deployment Manager will deploy resources in parallel. You can even abstract parts of your configuration into individual building blocks or templates that can be used for other configurations.

Deployment Manager uses the underlying APIs of each GCP service to deploy your resources. This enables you to deploy almost everything we have seen so far, from instances, instance templates, and groups, to VPC networks, firewall rules, VPN tunnels, Cloud Routers, and load balancers. For a full list of supported resource types, see the links section of this video: [https://cloud.google.com/deployment-manager/docs/configuration/supported-resource-types]



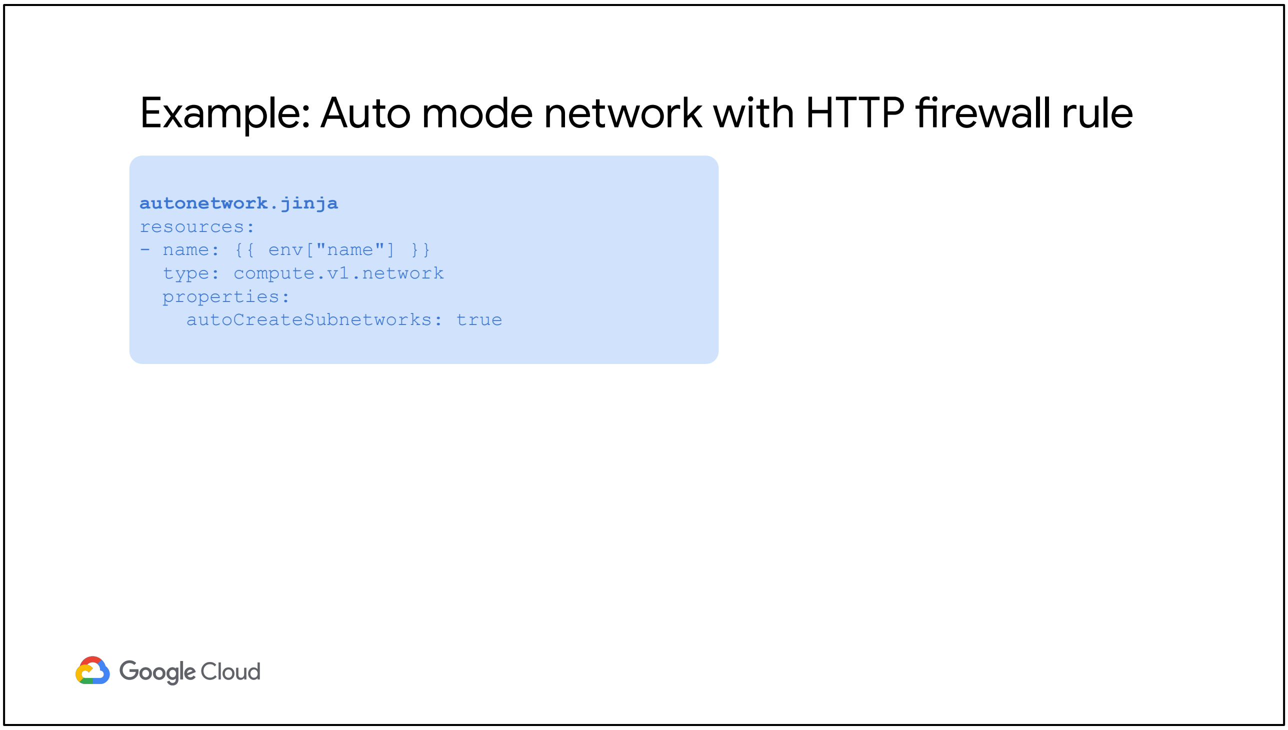
Example: Auto mode network with HTTP firewall rule

Auto mode network

HTTP Firewall Rule

Before you get into the lab, let me walk you through a quick example that shows how Deployment Manager can be used to set up an auto mode network with an HTTP firewall rule.

I could put this whole deployment into one single configuration; however, it's useful to parameterize your configuration with templates. Specifically, we are going to create one template for the auto mode network and one for the firewall rule. Therefore, if we want to create either of these resources somewhere else later on, we can use those templates.

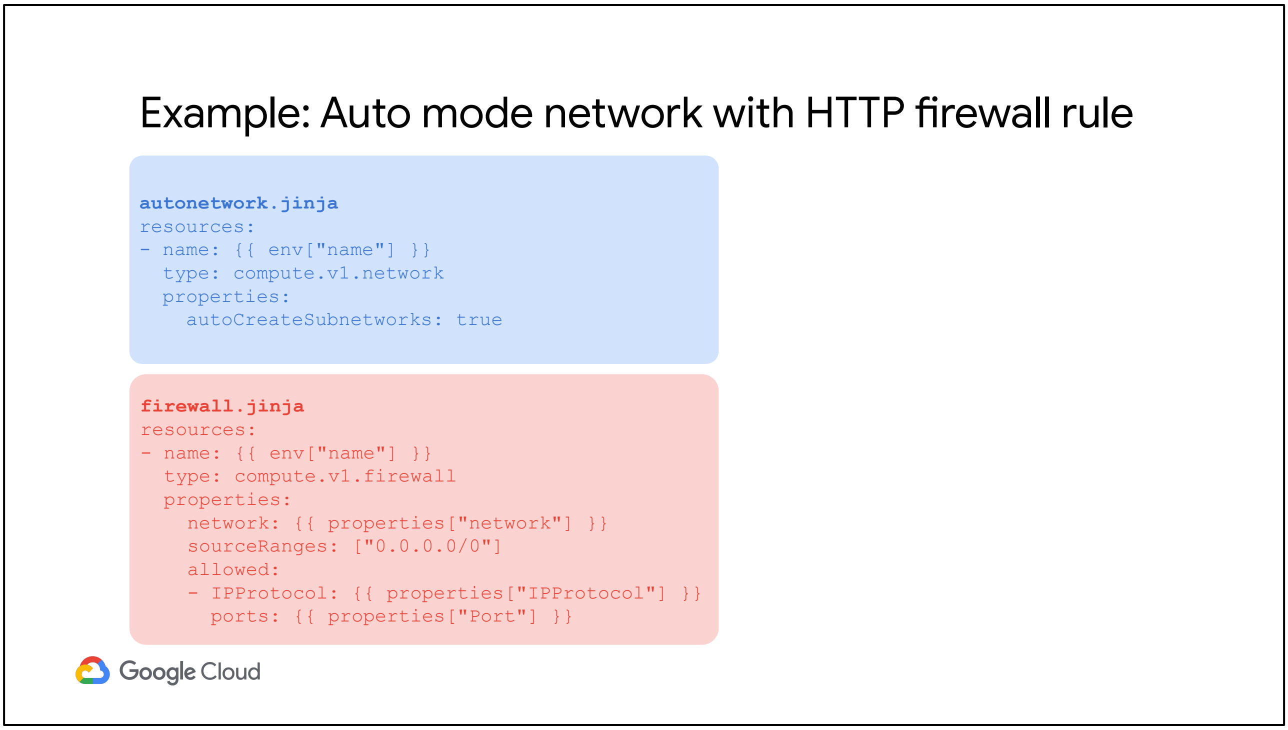


Example: Auto mode network with HTTP firewall rule

```
autonetwork.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.network
  properties:
    autoCreateSubnetworks: true
```

Let's start with the auto mode network template, which we can write in Jinja2 or Python. Now, each resource must contain a name, type, and properties:

- For the name, I am using an environment variable to get the name from the top-level configuration, which makes this template more flexible.
- For the type, I am defining the API for a VPC network, which is compute.v1.network. You can find all supported types in the documentation or query them within Cloud Shell, as you will explore in the upcoming lab.
- By definition, an auto mode network automatically creates a subnetwork in each region. Therefore, I am setting the autoCreateSubnetworks property to true.



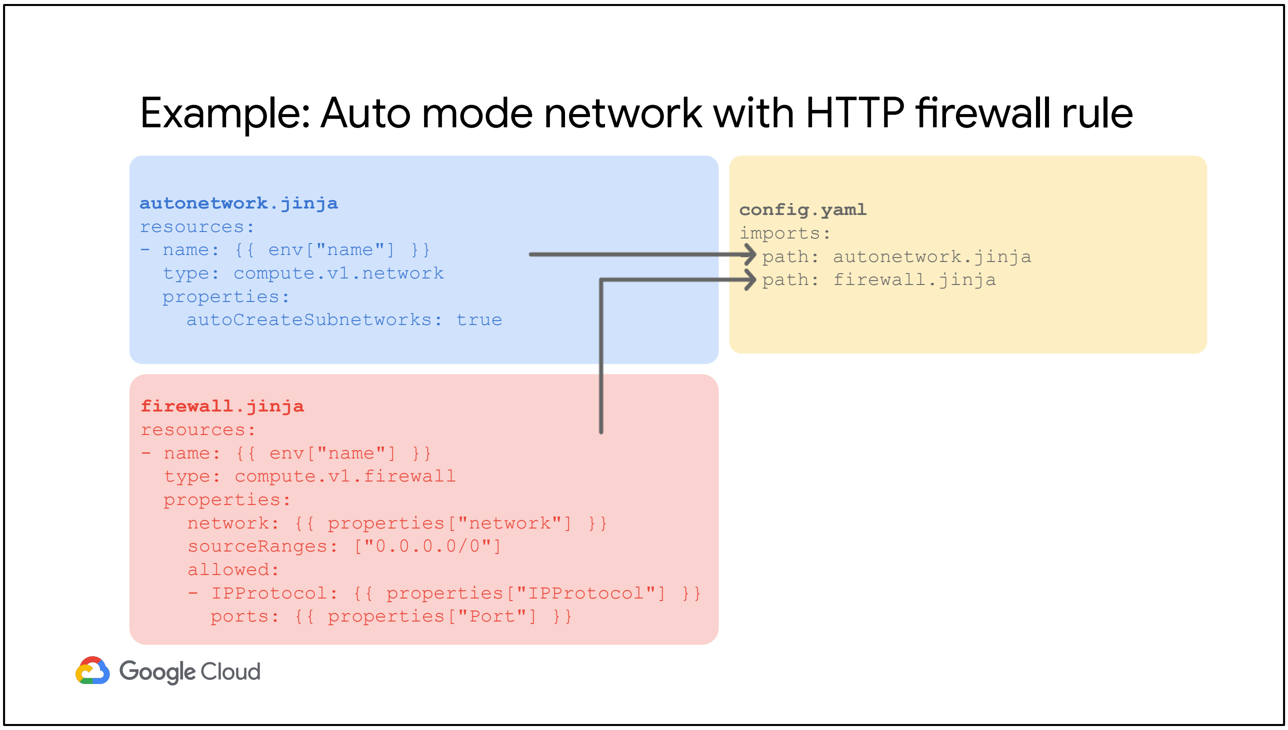
Example: Auto mode network with HTTP firewall rule

```
autonetwork.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.network
  properties:
    autoCreateSubnetworks: true
```

```
firewall.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.firewall
  properties:
    network: {{ properties["network"] }}
    sourceRanges: ["0.0.0.0/0"]
    allowed:
    - IPProtocol: {{ properties["IPProtocol"] }}
      ports: {{ properties["Port"] }}
```

Next, let's write the template for the HTTP firewall rule:

- For the name, I am again using an environment variable to get the name from the top-level configuration.
- For the type, I am defining the API for a firewall rule, which is compute.v1.firewall.
- The properties section contains the network I want to apply this firewall rule to, the source IP ranges, and the protocols and ports that are allowed. Except for the source IP ranges, I am defining these properties as template properties. I will provide the exact properties from the top-level configuration, which makes this firewall rule extremely flexible. Essentially, I can use this firewall rule template for any network and any protocol and port combination.



Example: Auto mode network with HTTP firewall rule

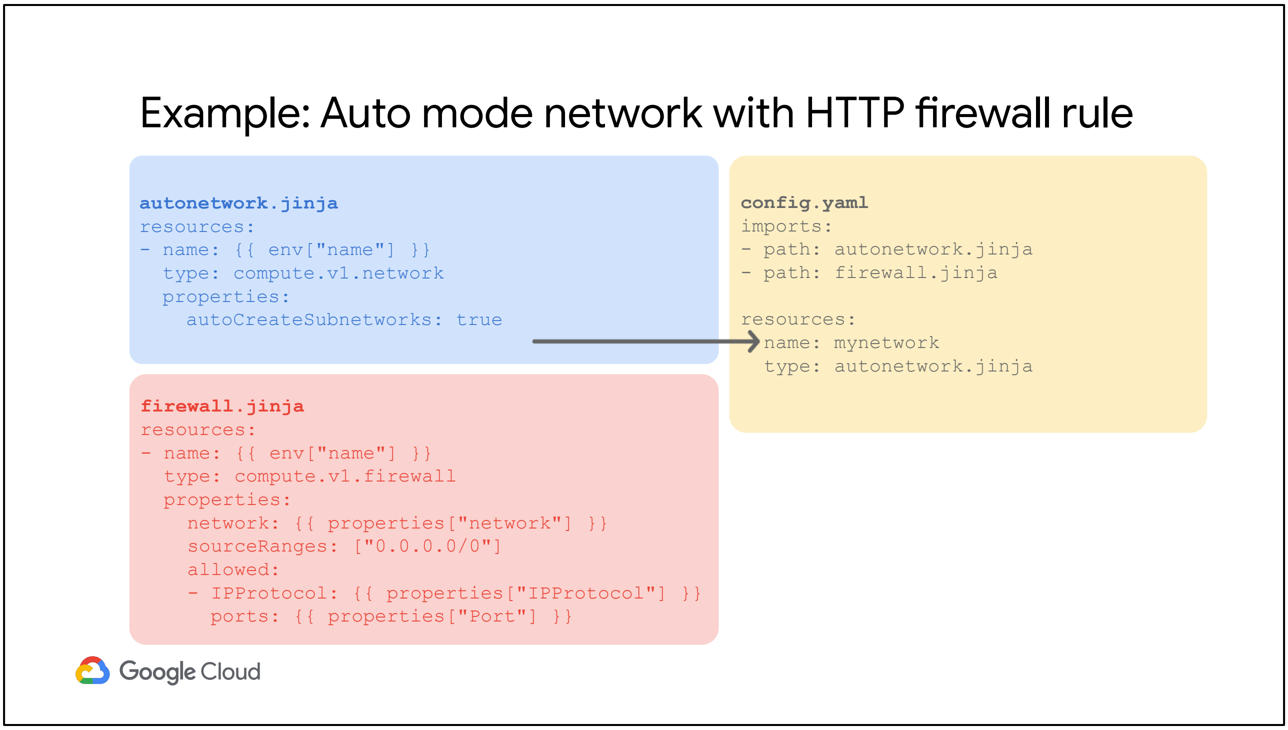
```
autonetwork.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.network
  properties:
    autoCreateSubnetworks: true
```

```
config.yaml
imports:
  path: autonetwork.jinja
  path: firewall.jinja
```

```
firewall.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.firewall
  properties:
    network: {{ properties["network"] }}
    sourceRanges: ["0.0.0.0/0"]
    allowed:
    - IPProtocol: {{ properties["IPProtocol"] }}
      ports: {{ properties["Port"] }}
```

Next, let’s write the top-level configuration in YAML syntax.

I start by importing the templates that I want to use in this configuration, which are autonetwork.jinja and firewall.jinja.



Example: Auto mode network with HTTP firewall rule

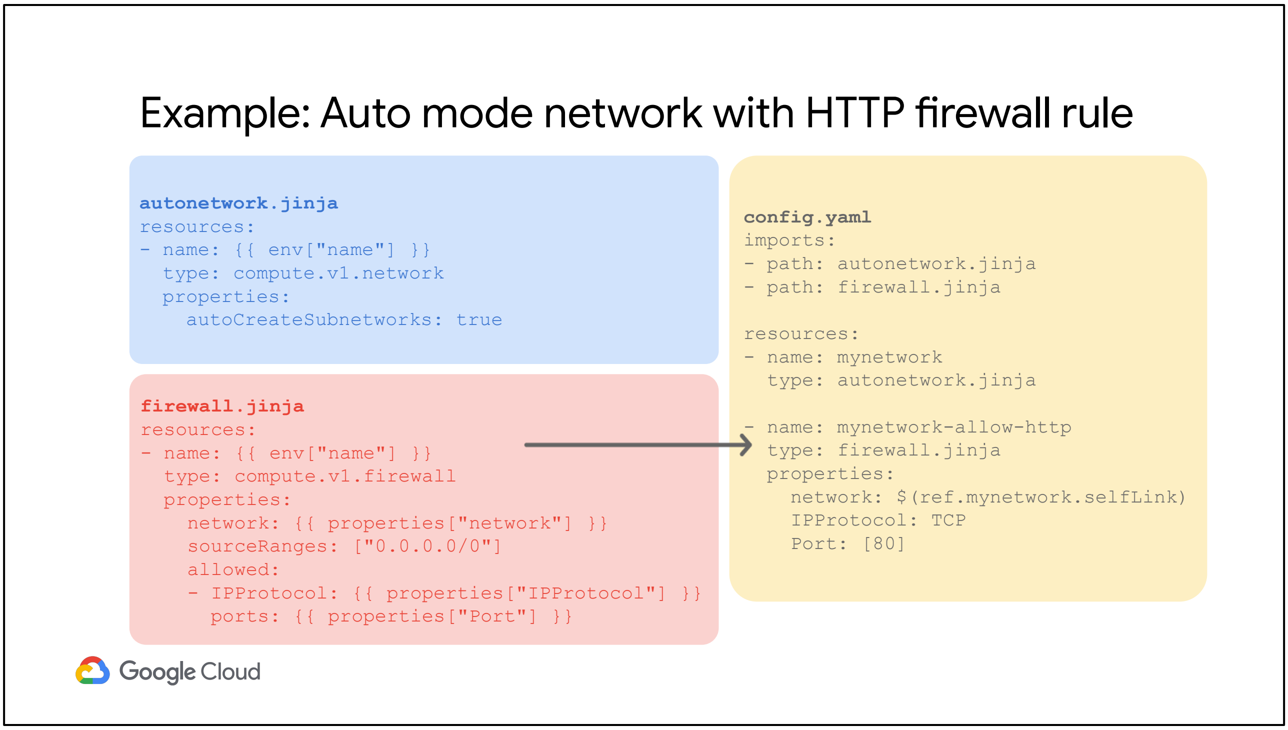
```
autonetwork.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.network
  properties:
    autoCreateSubnetworks: true
```

```
firewall.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.firewall
  properties:
    network: {{ properties["network"] }}
    sourceRanges: ["0.0.0.0/0"]
    allowed:
    - IPProtocol: {{ properties["IPProtocol"] }}
      ports: {{ properties["Port"] }}
```

```
config.yaml
imports:
- path: autonetwork.jinja
- path: firewall.jinja

resources:
  name: mynetwork
  type: autonetwork.jinja
```

Then, I define the auto mode network by giving it the name “mynetwork” and leveraging the autonetwork.jinja template. I could create more auto mode networks in this configuration with other names, or simply reuse this template in other configurations, later on.



Example: Auto mode network with HTTP firewall rule

```
autonetwork.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.network
  properties:
    autoCreateSubnetworks: true
```

```
firewall.jinja
resources:
- name: {{ env["name"] }}
  type: compute.v1.firewall
  properties:
    network: {{ properties["network"] }}
    sourceRanges: ["0.0.0.0/0"]
    allowed:
    - IPProtocol: {{ properties["IPProtocol"] }}
      ports: {{ properties["Port"] }}
```

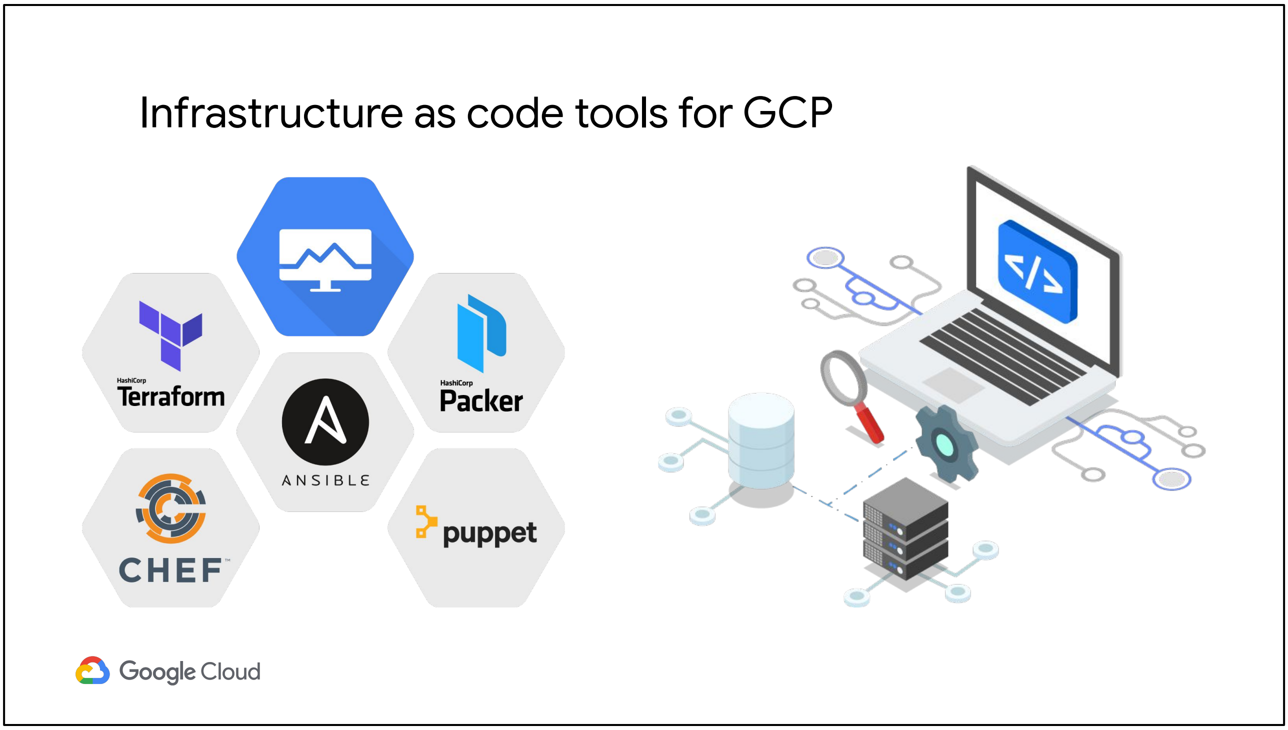
```
config.yaml
imports:
- path: autonetwork.jinja
- path: firewall.jinja

resources:
- name: mynetwork
  type: autonetwork.jinja

- name: mynetwork-allow-http
  type: firewall.jinja
  properties:
    network: $(ref.mynetwork.selfLink)
    IPProtocol: TCP
    Port: [80]
```

Now, I define the firewall rule, by giving it a name, leveraging the firewall.jinja template, referencing mynetwork, and defining the IP protocol and port. I can easily add other ports, such as 443 for HTTPS or 22 for SSH traffic.

Using the selfLink reference for the network name ensures that the VPC network is created before the firewall rule. This is very important because Deployment Manager creates all the resources in parallel, unless you use references. You would get an error without the reference because you cannot create a firewall rule for a non-existing network.

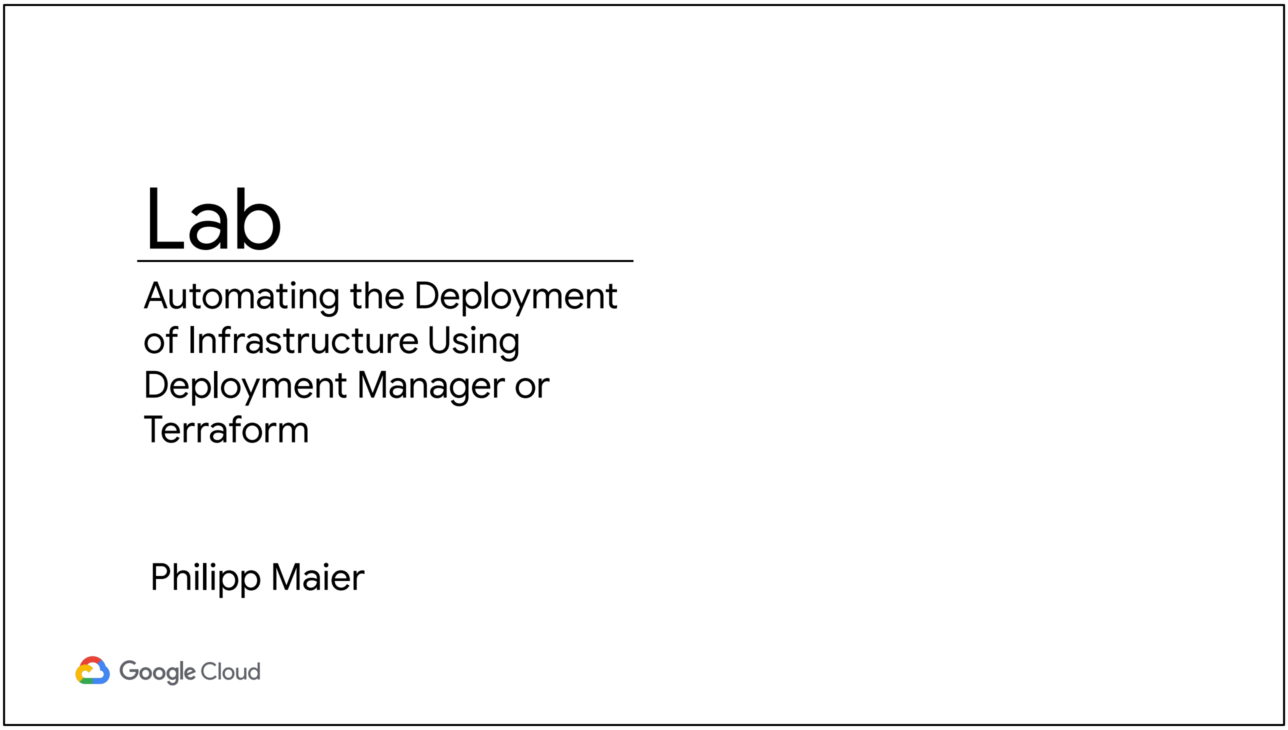


Infrastructure as code tools for GCP

Now, there are other infrastructure automation tools in addition to Deployment Manager that you can use in GCP. You can also use Terraform, Chef, Puppet, Ansible, or Packer. All of these tools allow you to treat your infrastructure like software, which helps you decrease costs, reduce risks, and deploy faster by capturing infrastructure as code.

You might recognize some of these tools, because they work across many cloud service providers. I recommend that you provision and manage resources on Google Cloud with the tools you already know. That's why, in the upcoming lab, you'll have the choice of using Deployment Manager or Terraform to automate the deployment of infrastructure.

For more information on each of these tools, see the links section of this video: [https://cloud.google.com/solutions/infrastructure-as-code/#cards]



Lab

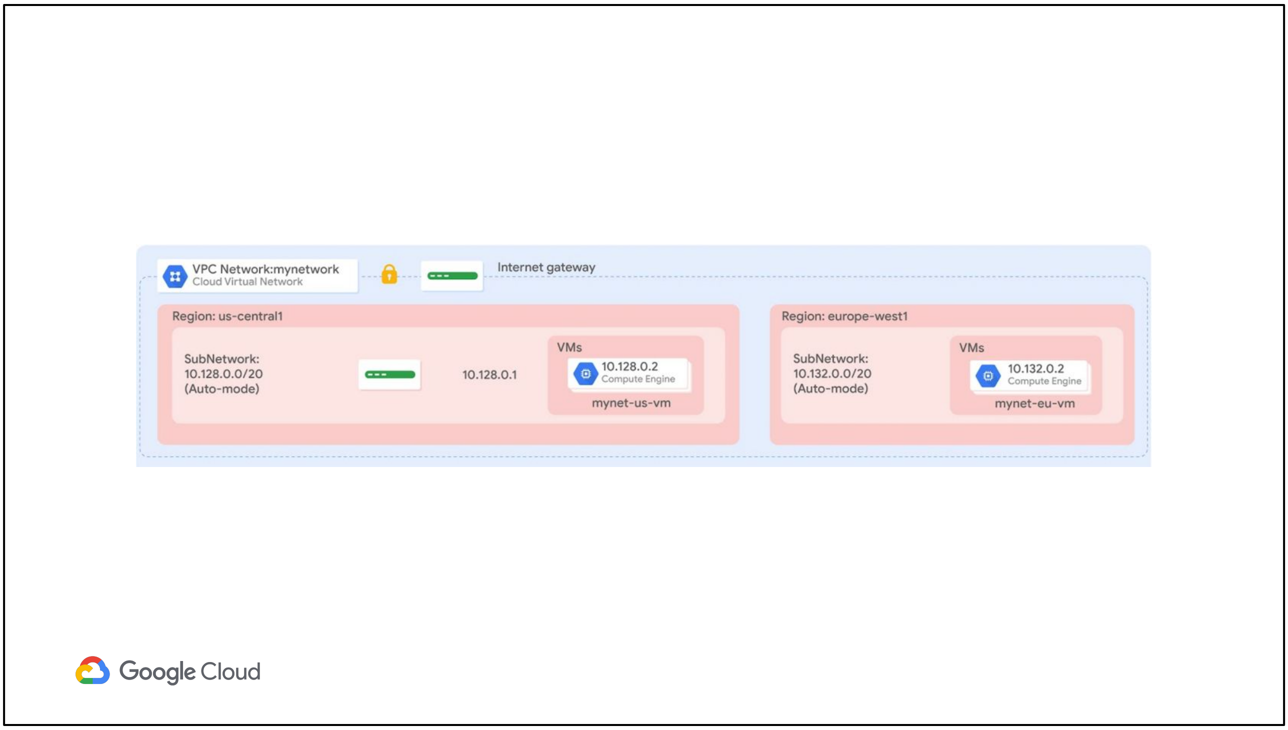
Automating the Deployment of Infrastructure Using Deployment Manager or Terraform

Philipp Maier

Google Cloud

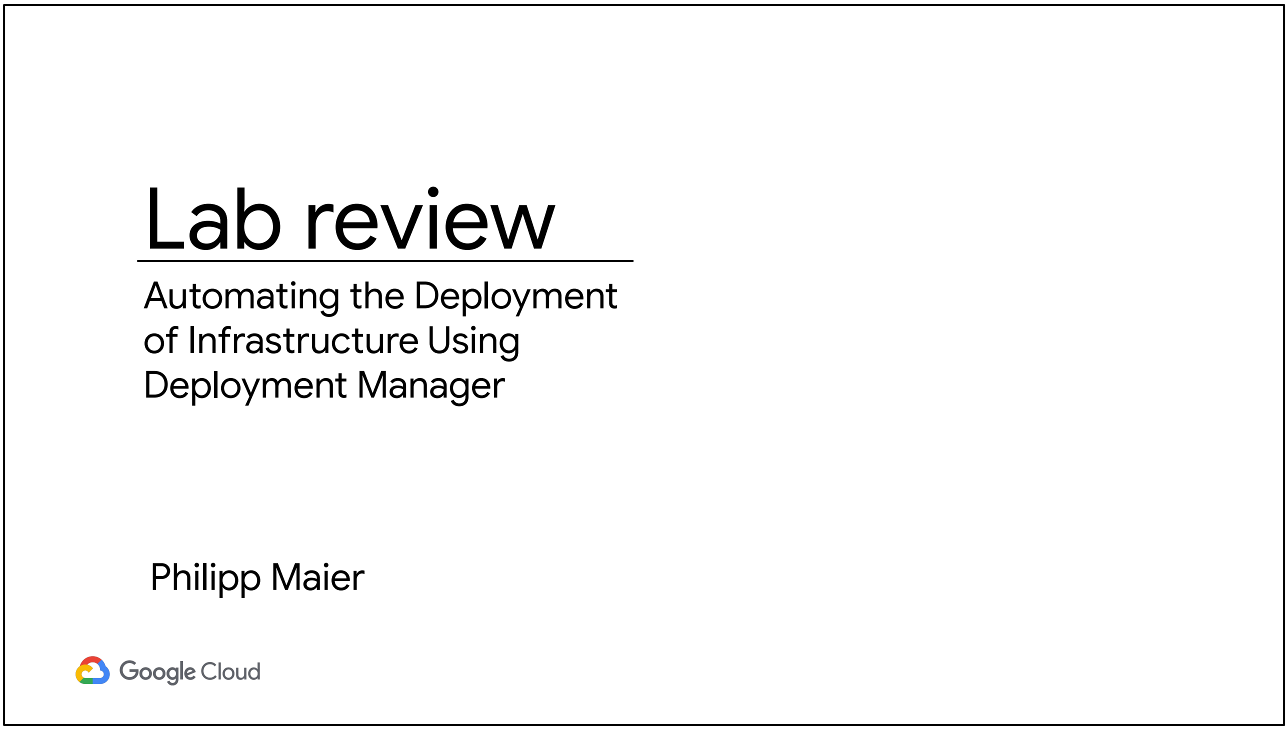
Let’s apply what we just covered to automate the deployment of VPC networks, firewall rules, and VM instances.

In this lab, we are giving you two choices: you can deploy resources by using Deployment Manager or Terraform.



Either way, you deploy an auto mode network called mynetwork with a firewall rule to allow HTTP, SSH, RDP, and ICMP traffic. You also deploy the VM instances shown in this network diagram.

Again, you have the choice of deploying these resources with Deployment Manager or Terraform through two separate labs. You can complete either lab or even both!



Lab review

Automating the Deployment of Infrastructure Using Deployment Manager

Philipp Maier

Google Cloud

In this lab, you created a Deployment Manager configuration and template to automate the deployment of GCP infrastructure. Templates can be very flexible because of their environment and template variables. Therefore, the benefit of creating templates is that they can be reused across many configurations.

You can use the template that you created as a starting point for future deployments.

You can stay for a lab walkthrough, but remember that GCP's user interface can change, so your environment might look slightly different.

Lab review

Automating the Deployment of Infrastructure Using Terraform

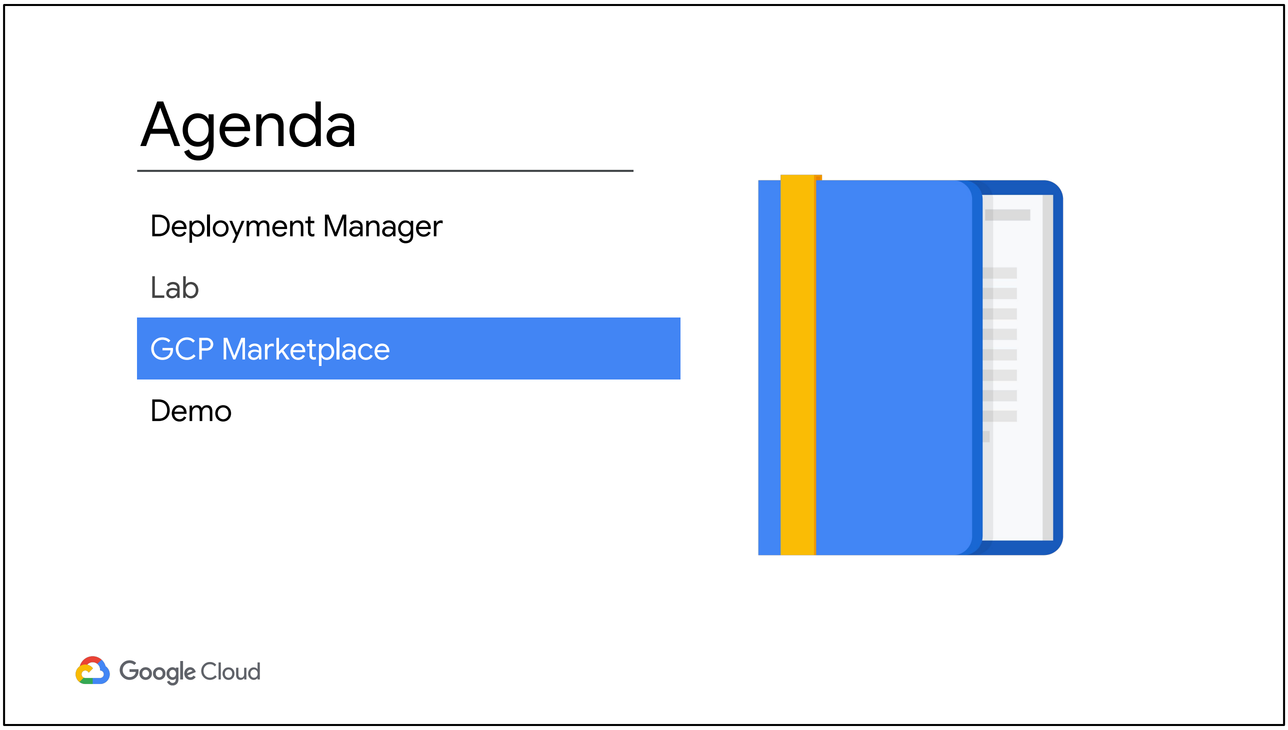
Philipp Maier

Google Cloud

In this lab, you created a Terraform configuration with a module to automate the deployment of GCP infrastructure. As your configuration changes, Terraform can create incremental execution plans, which allows you to build your overall configuration step by step.

The instance module allowed you to re-use the same resource configuration for multiple resources while providing properties as input variables. You can leverage the configuration and module that you created as a starting point for future deployments.

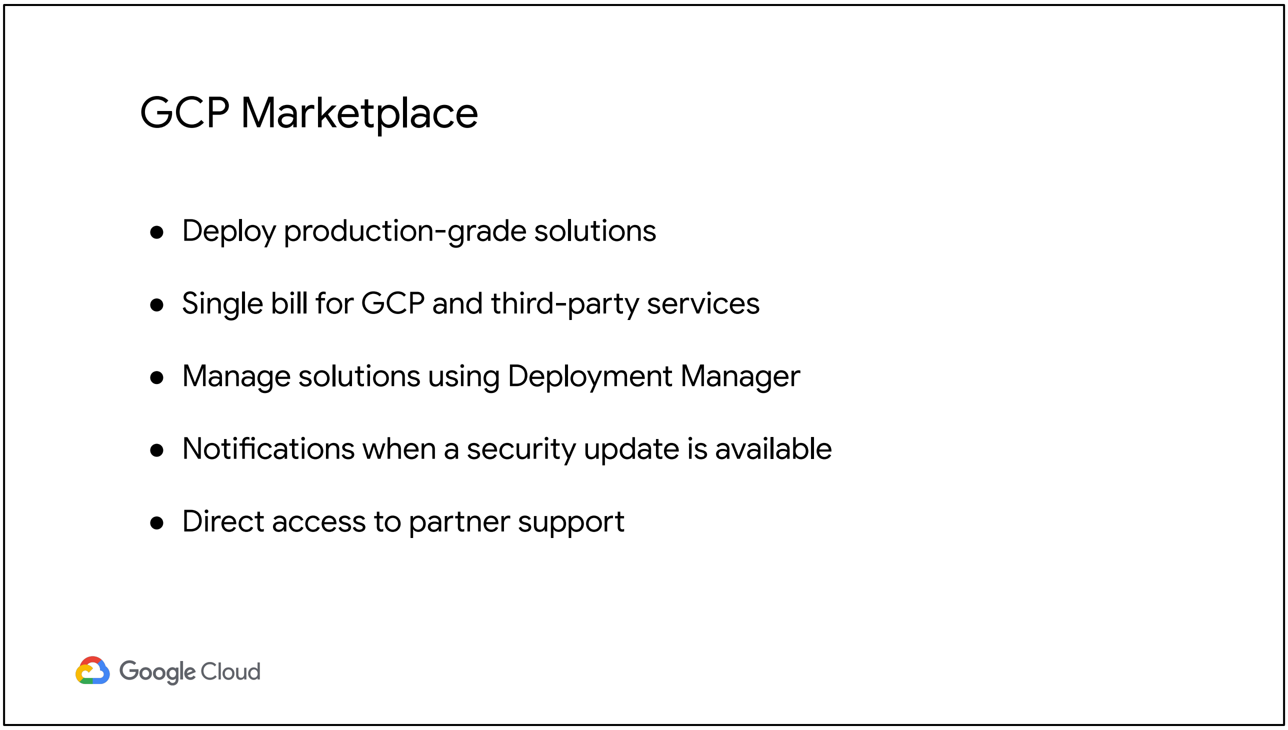
You can stay for a lab walkthrough, but remember that GCP's user interface can change, so your environment might look slightly different.



Agenda

- Deployment Manager
- Lab
- GCP Marketplace
- Demo

Let’s learn a little more about GCP Marketplace.



GCP Marketplace

- Deploy production-grade solutions
- Single bill for GCP and third-party services
- Manage solutions using Deployment Manager
- Notifications when a security update is available
- Direct access to partner support

Google Cloud

GCP Marketplace lets you quickly deploy functional software packages that run on GCP. Essentially, GCP Marketplace offers production-grade solutions from third-party vendors who have already created their own deployment configurations based on Deployment Manager. These solutions are billed together with all of your project's GCP services. If you already have a license for a third-party service, you might be able to use a Bring Your Own License solution.

You can deploy a software package now and scale that deployment later when your applications require additional capacity. GCP even updates the images of these software packages to fix critical issues and vulnerabilities, but doesn't update software that you have already deployed**.** You even get direct access to partner support.

Demo

Launch on GCP Marketplace

Philipp Maier

Let me show you how to launch infrastructure solutions on GCP Marketplace.
My goal is to deploy a LAMP stack to a single Compute Engine instance. A LAMP stack consists of Linux, Apache HTTP Server, MySQL, and PHP.

[DEMO]

That’s how it easy it is to launch infrastructure solutions on GCP Marketplace.

Review

Infrastructure Automation

Google Cloud

In this module, we automated the deployment of infrastructure using Deployment Manager and Terraform, and launched infrastructure solutions using GCP Marketplace.

Now, you might say that going through all the effort to deploy a network, a firewall rule, and two VM instances doesn’t convince you to use Deployment Manager or Terraform.

That’s true, if you only need to create these resources once, and don’t foresee ever creating them again. However, for those of us who manage several resources and need to deploy, update, and destroy them in a repeatable way, infrastructure automation tools like Deployment Manager and Terraform become essential.