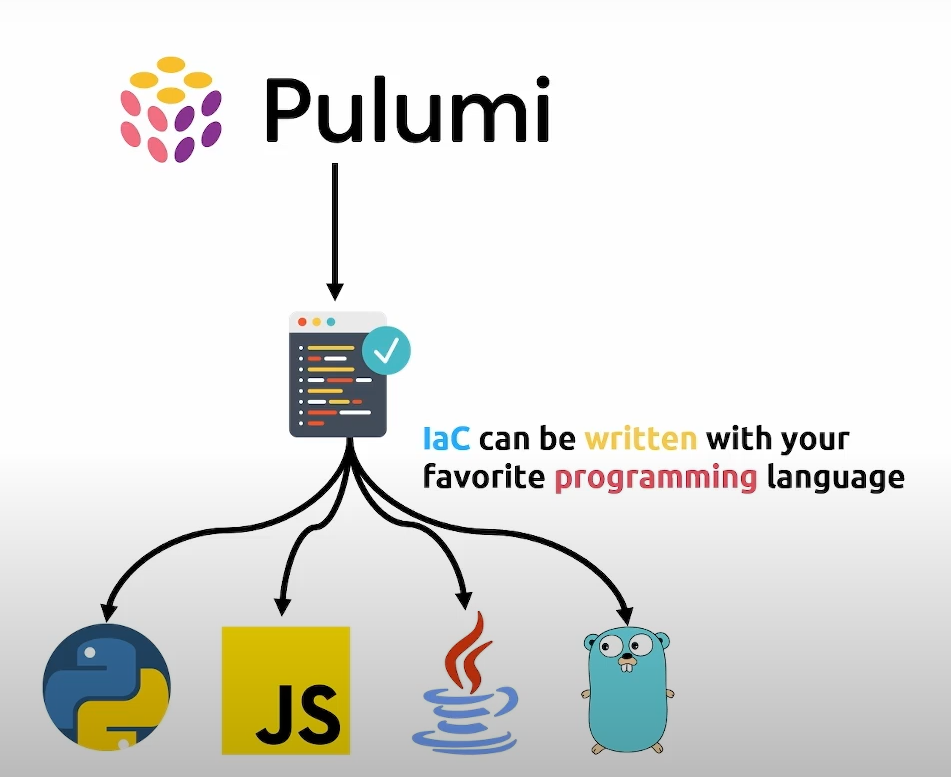
**Pulumi: Simplified Overview**

#### **What is Pulumi?**

Pulumi is an open-source Infrastructure as Code (IaC) platform that lets you define, deploy, and manage cloud infrastructure using modern programming languages like Python, TypeScript, Go, C#, and Java. It supports all major cloud providers (AWS, Azure, GCP, Kubernetes, etc.) and on-premises environments, making it a versatile and powerful tool for automating, securing, and managing your infrastructure.



**Pulumi Features:**

1. **Languages You Love**:

Pulumi allows you to model cloud infrastructure using languages you are already familiar with, including TypeScript, JavaScript, Python, Go, .NET, Java, and YAML. It takes advantage of the features of each language, enabling a flexible and familiar development experience.

1. **Build on Any Cloud**:
   1. Pulumi provides full access to a wide range of services across major cloud providers such as AWS, Azure, GCP, and more than 60 additional providers. With a consistent SDK interface, you can interact with all cloud resources seamlessly.
2. **Create Reusable Infrastructure**:
   1. You can build and share reusable cloud infrastructure using Pulumi Packages. These packages encapsulate your cloud architectures, best practices, and configurations, making it easier to standardize and share across teams or projects.
3. **Multiple Deployment Options**:
   1. Pulumi supports a variety of deployment methods. You can deploy infrastructure interactively through the CLI, programmatically with the Pulumi Automation API, or integrate Pulumi into your CI/CD pipelines for automated deployments.
4. **Deployments as Code**:
   1. With the Automation API, you can run infrastructure deployments directly from your application code at runtime. This opens up possibilities for building infrastructure APIs, custom platforms, and command-line interfaces (CLIs) that interact with cloud resources programmatically.
5. **Import Code from Any IaC Platform**:
   1. Pulumi enables you to import existing infrastructure code from other Infrastructure-as-Code (IaC) platforms like Terraform. Pulumi handles all dependencies, making it easier to manage and convert infrastructure from other tools.
6. **Code Converters for Seamless Transition**:
   1. One of Pulumi's standout features is its code converters. You can import infrastructure code from various platforms and convert it into your preferred programming language. This allows for a smooth transition and adoption of Pulumi, even if you're migrating from other IaC tools.

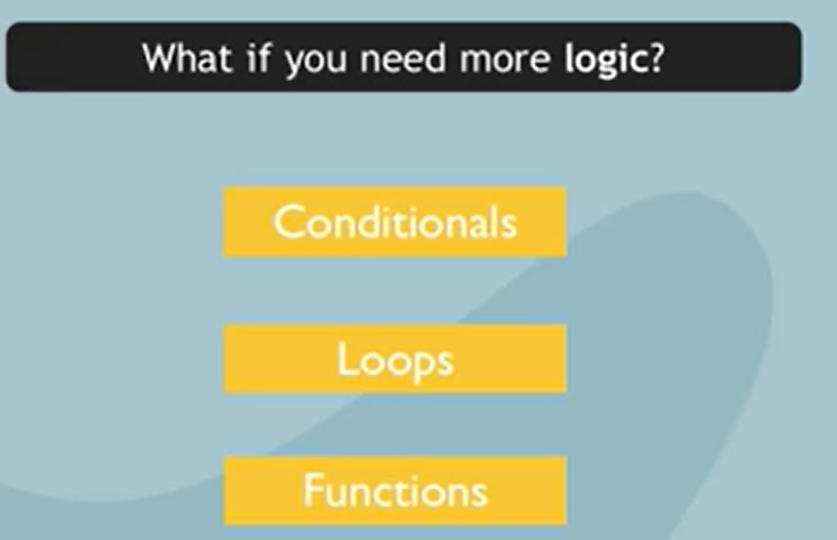
## **One Pipeline for Everyone**



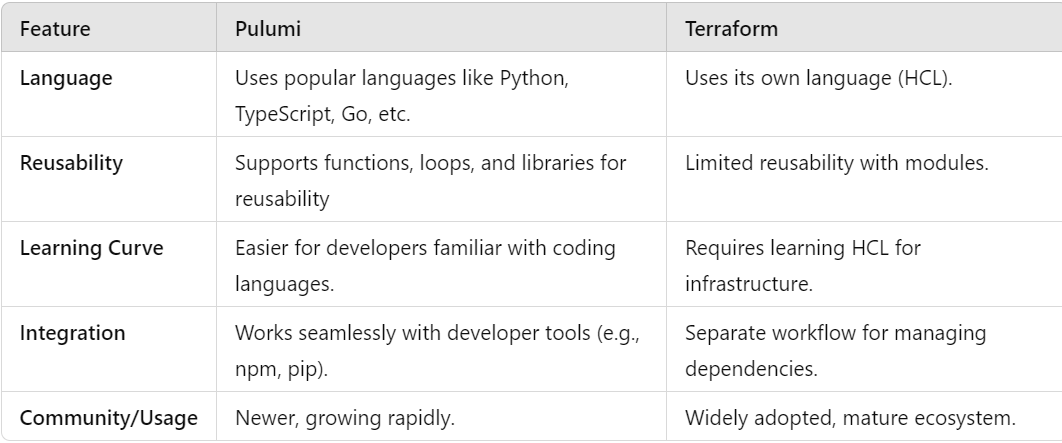
Pulumi lets you use one pipeline for Application code and Infrastructure code. Pulumi’s approach to infrastructure as code is great for continuous delivery because it uses source code to model cloud resources. This means updates to your cloud infrastructure can be reviewed, validated, and tested using the same process that you have today. For example, doing code reviews via Pull Requests, running code through linters or static analysis tools, and running unit and integration tests as appropriate. It all “just works” for your cloud infrastructure the same way it would for your application code.

Pulumi can easily integrate into any continuous integration/continuous delivery (CI/CD) system.

### **When to Use Pulumi Over Terraform**



* **Dynamic and complex infrastructure needs**.
* **Multi-cloud environments** with diverse workloads.
* Teams with **strong programming expertise**.
* Projects requiring **Kubernetes management** alongside other resources.
* Organizations that prioritize **security policies** and **developer experience**.



### **Pulumi Example: Using Loops for Reusability**

Suppose you want to create multiple virtual machines in Pulumi. Instead of writing repetitive code, you can use a loop.

|  |
| --- |
| import pulumi  from pulumi\_gcp import compute    # Define a list of VM names  vm\_names = ["vm1", "vm2", "vm3"]    # Create multiple VMs dynamically using a loop  for name in vm\_names:  compute.Instance(  f"{name}-instance",  machine\_type="e2-micro",  zone="us-central1-a",  boot\_disk={  "initializeParams": {"image": "debian-cloud/debian-10"},  },  network\_interfaces=[{  "network": "default",  "accessConfigs": [{}],  }]  ) |

### **Terraform Example: Using Modules for Reusability**

In Terraform, you’d create a reusable **module** for a virtual machine and call it multiple times with different variables.

**Module Code (e.g., modules/vm/main.tf):**

|  |
| --- |
| resource "google\_compute\_instance" "vm" {  name = var.vm\_name  machine\_type = "e2-micro"  zone = "us-central1-a"    boot\_disk {  initialize\_params {  image = "debian-cloud/debian-10"  }  }    network\_interface {  network = "default"  access\_config {}  }  } |

**Main Code (e.g., main.tf):**

|  |
| --- |
| module "vm1" {  source = "./modules/vm"  vm\_name = "vm1"  }    module "vm2" {  source = "./modules/vm"  vm\_name = "vm2"  }    module "vm3" {  source = "./modules/vm"  vm\_name = "vm3"  } |

**Install Pulumi CLI**

**Download the Pulumi CLI**

* **Windows:** Download the installer from Pulumi's official website or use Chocolatey:

|  |
| --- |
| choco install pulumi |

* **Linux/macOS:** Use the following script:

|  |
| --- |
| curl -fsSL <https://get.pulumi.com> | sh |

* Add Pulumi to your PATH if it is not automatically added.

**Verify installation**

|  |
| --- |
| pulumi version |

This will confirm Pulumi is installed correctly.

### **Install Python**

1. Ensure Python (3.7 or later) is installed on your system:
   1. **Linux/macOS:** Use your package manager:

|  |
| --- |
| sudo apt install python3 python3-pip # Ubuntu/Debian brew install python # macOS |

* 1. **Windows:** Download and install Python from [python.org](https://www.python.org/).

1. Verify the installation:

|  |
| --- |
| python --version pip --version |

### **Install Pulumi's Python SDK**

1. **Install the Pulumi Python SDK using pip:**

|  |
| --- |
| pip install pulumi |

1. **Verify the installation:** Check that pulumi is available in your virtual environment:

|  |
| --- |
| pulumi version |

### **Configure Pulumi**

1. **Log in to Pulumi:** Pulumi requires a backend for storing your state files. Run:

|  |
| --- |
| pulumi login |

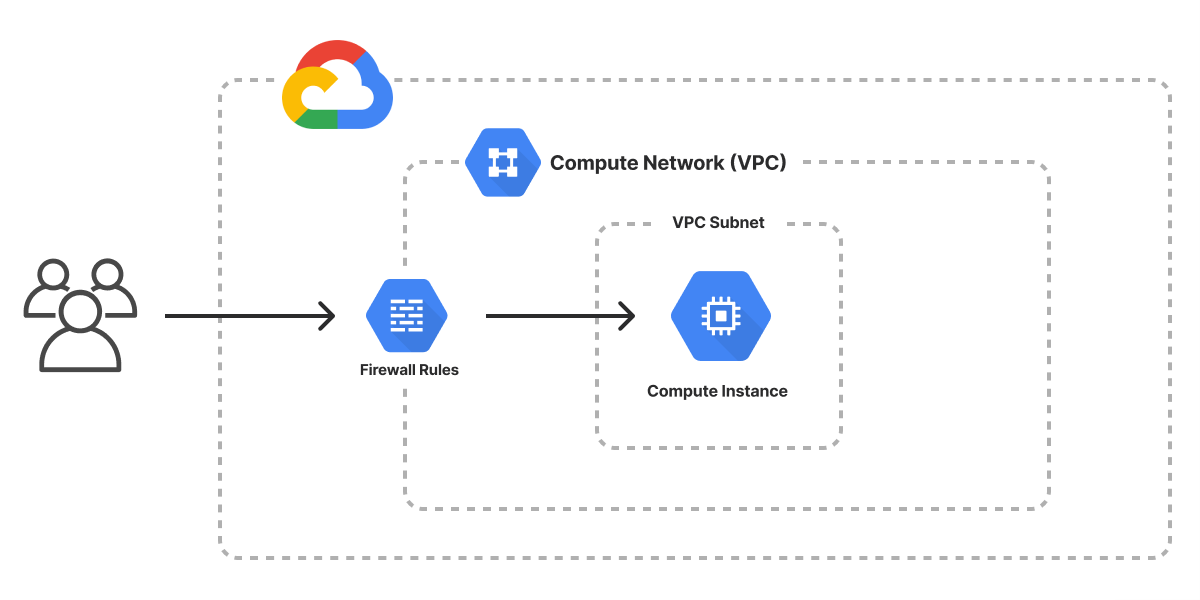
Choose one of the following options:

* 1. **Pulumi Service:** Logs you into Pulumi's managed backend.
  2. **Local:** Uses your local filesystem for state storage:

|  |
| --- |
| pulumi login --local |

**Create a New Pulumi Project**

# **Virtual Machine on Google Cloud**



**Create a directory for your project**:

|  |
| --- |
| mkdir my-pulumi-gcp-project cd my-pulumi-gcp-project |

**Initialize a new Pulumi project for Python**:

|  |
| --- |
| pulumi new gcp-python |

Follow the prompts to configure your project:

* + **Project Name**: my-gcp-project
  + **Stack Name**: dev
  + **Project Description**: A GCP project with VPC, firewall rules, and compute instance

### **Install Pulumi Libraries for GCP**

Install the Pulumi library for GCP:

|  |
| --- |
| pip install pulumi-gcp |

### **Write Your Pulumi Program**

1. Open the \_\_main\_\_.py file in the my-pulumi-gcp-project directory.
2. Define your infrastructure. Here's an example that follows the diagram: Create a VPC network, a subnet, a firewall rule, and a Compute Engine instance in the VPC.

\_\_main\_\_.py

|  |
| --- |
| import pulumi  import pulumi\_gcp as gcp  # Define variables  project\_id = "devops-project-439310" # Replace with your GCP project ID  zone = "us-central1-a"  instance\_name = "pulumi-vm"  # Step 1: Create a VPC network  vpc\_network = gcp.compute.Network("my-vpc", auto\_create\_subnetworks=False)  # Step 2: Create a VPC subnet  subnet = gcp.compute.Subnetwork(  "my-subnet",  network=vpc\_network.name,  ip\_cidr\_range="10.0.0.0/24",  region="us-central1"  )  # Step 3: Create a firewall rule to allow SSH access  firewall\_rule = gcp.compute.Firewall(  "allow-ssh",  network=vpc\_network.name,  allows=[gcp.compute.FirewallAllowArgs(  protocol="tcp",  ports=["22"]  )],  source\_ranges=["0.0.0.0/0"], # Allow from all IPs  target\_tags=["ssh-access"]  )  # Step 4: Create the VM instance with the network and firewall  vm\_instance = gcp.compute.Instance(  instance\_name,  name=instance\_name,  machine\_type="e2-micro", # Change to your desired machine type  zone=zone,  boot\_disk=gcp.compute.InstanceBootDiskArgs(  initialize\_params=gcp.compute.InstanceBootDiskInitializeParamsArgs(  image="projects/debian-cloud/global/images/family/debian-12", # Choose your image  size=20, # Disk size in GB  ),  ),  network\_interfaces=[gcp.compute.InstanceNetworkInterfaceArgs(  network=vpc\_network.name,  subnetwork=subnet.name,  access\_configs=[gcp.compute.InstanceNetworkInterfaceAccessConfigArgs()],  )],  tags=["ssh-access"]  )  # Export the instance's external IP  pulumi.export("instance\_ip", vm\_instance.network\_interfaces[0].access\_configs[0].nat\_ip) |

### **Configure GCP Credentials**

Pulumi uses your GCP credentials to interact with your GCP account.

1. **Authenticate with GCP**:

|  |
| --- |
| gcloud auth application-default login |

1. **Set your GCP project ID**:

|  |
| --- |
| gcloud config set project [YOUR\_PROJECT\_ID] |

|  |
| --- |
| pulumi config set gcp:project devops-project-439310  pulumi config set gcp:zone us-central1-a |

### **Preview and Deploy Your Pulumi Program**

1. **Preview your changes**:

|  |
| --- |
| pulumi preview |

This will display the resources that Pulumi will create.

1. **Deploy your resources**:

|  |
| --- |
| pulumi up |

Confirm the deployment by typing "yes" when prompted.

### **Clean Up Resources (Optional)**

To destroy the resources created by Pulumi:

|  |
| --- |
| pulumi destroy |

**Conclusion**:

Pulumi simplifies cloud infrastructure management by allowing developers to use familiar programming languages. It supports multi-cloud environments and integrates seamlessly with CI/CD pipelines. This approach enhances flexibility, consistency, and scalability in managing cloud resources.