GeoPython 2023

Basel Switzerland March 06-08

title: Code-Centric Infrastructure as Code (IaC) using Pulumi with Python
speaker: Anmol Krishan Sachdeva
job: Hybrid Cloud Architect
company: Google
handle: @greatdevaks

Deck: tinyurl.com/geopython-pulumi-2023

GeoPython 2023

Basel Switzerland March 06-08

Follow Me

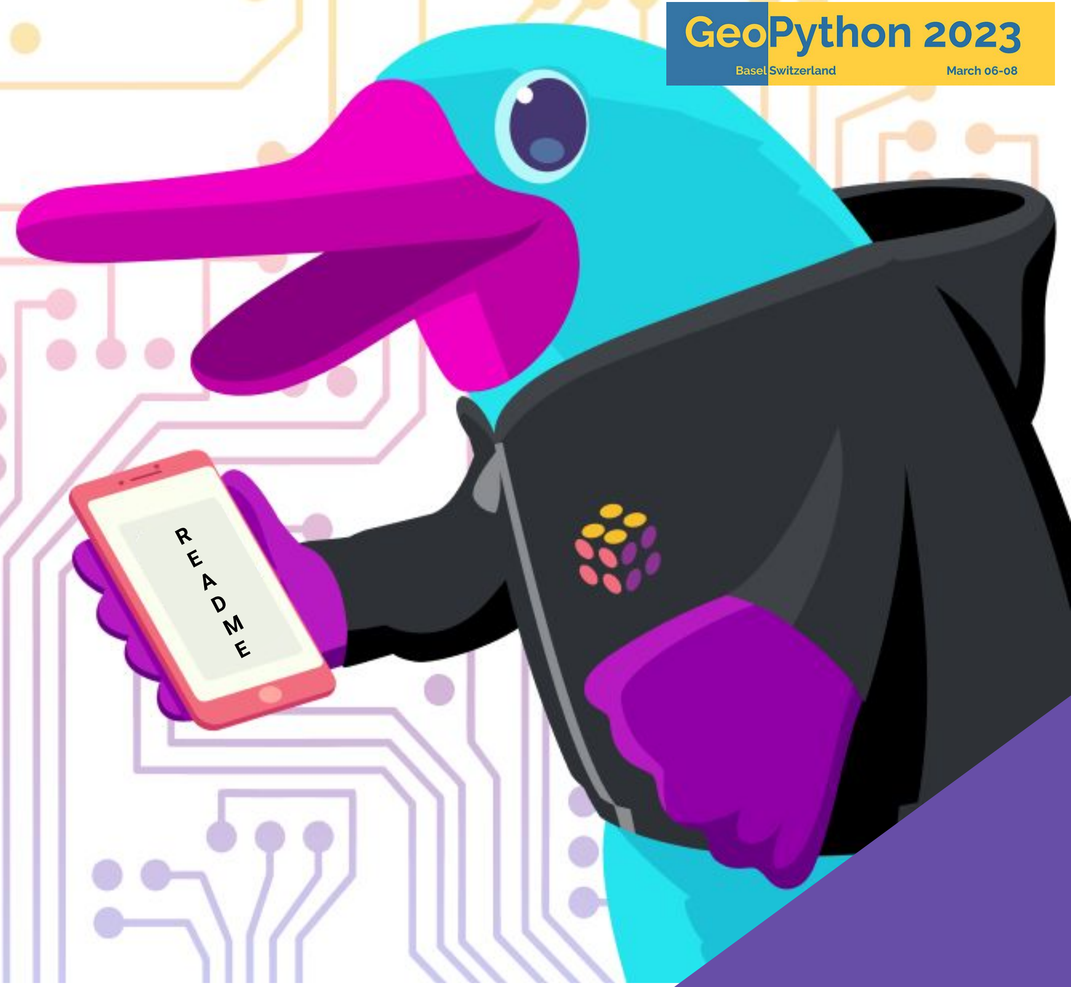
@greatdevaks

Pulumi

Code-Centric Infrastructure as Code (IaC) using Pulumi with Python

Anmol Krishan Sachdeva (@greatdevaks) | Hybrid Cloud Architect, Google

The content and views presented during the session are author’s own and not of any organizations they are associated with.

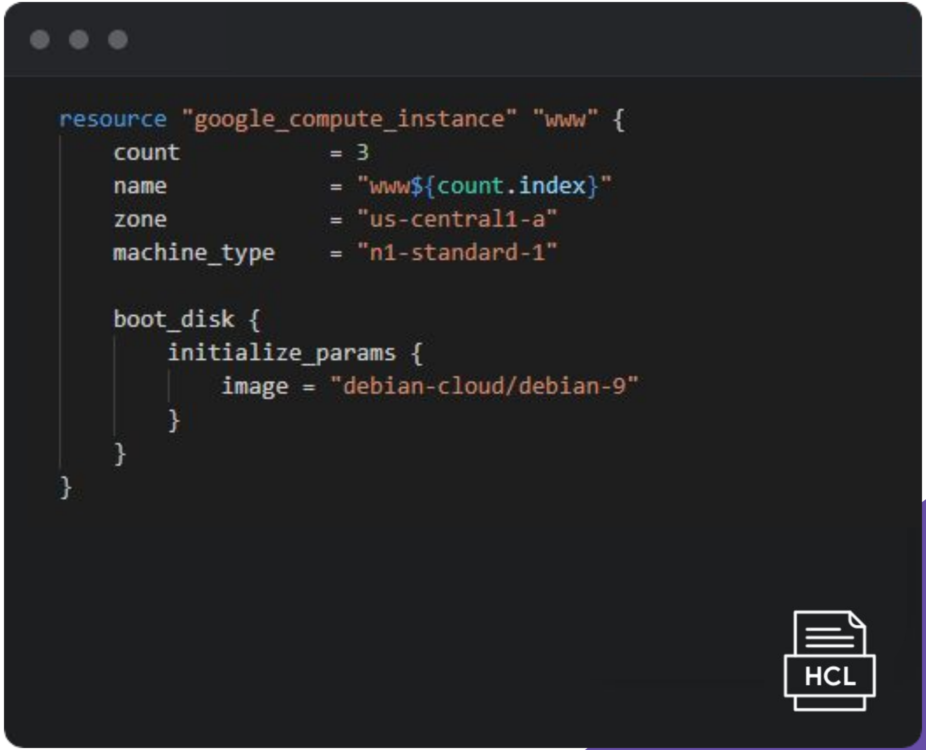


Agenda

- From Hardware to DSLs to General Purpose Language Code
- Introduction
- Imperative or Declarative?
- Terraform vs. Pulumi
- Architecture and Components
- Component Resources
- Pulumi Converters
- ***Get your hands PURPLE...***

From Hardware to DSLs

HashiCorp Terraform



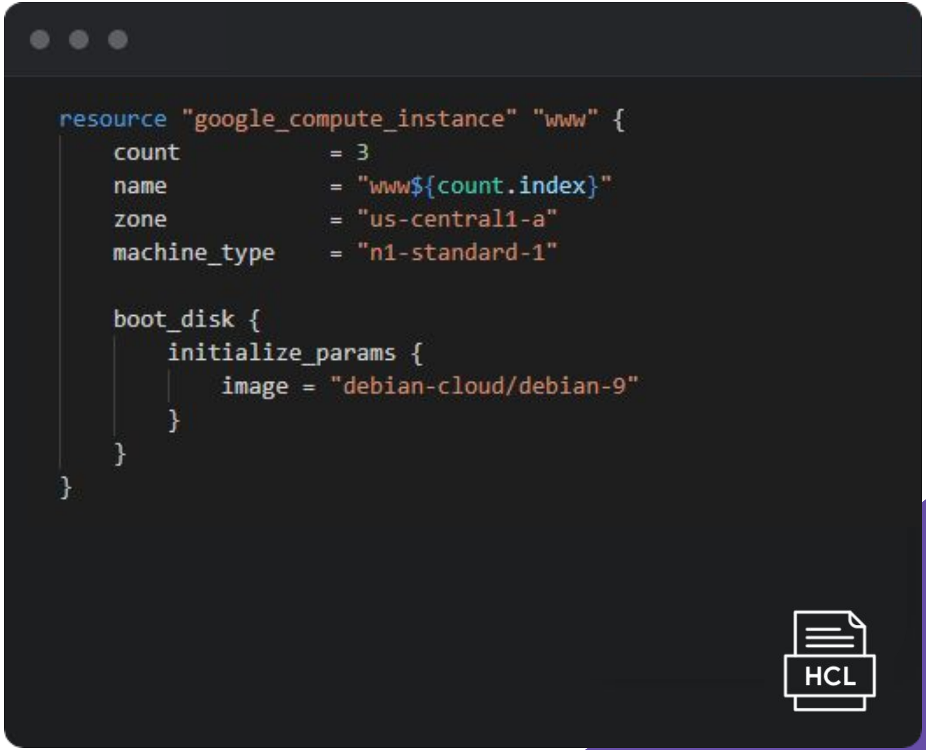
```
resource "google_compute_instance" "www" {
    count           = 3
    name            = "www${count.index}"
    zone            = "us-central1-a"
    machine_type    = "n1-standard-1"

    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-9"
        }
    }
}
```

From Hardware to DSLs

What if more **logic** and control is needed?

- Conditionals
- Loops
- Functions / Reuse
- Classes



```
resource "google_compute_instance" "www" {
    count            = 3
    name             = "www${count.index}"
    zone             = "us-central1-a"
    machine_type     = "n1-standard-1"

    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-9"
        }
    }
}
```

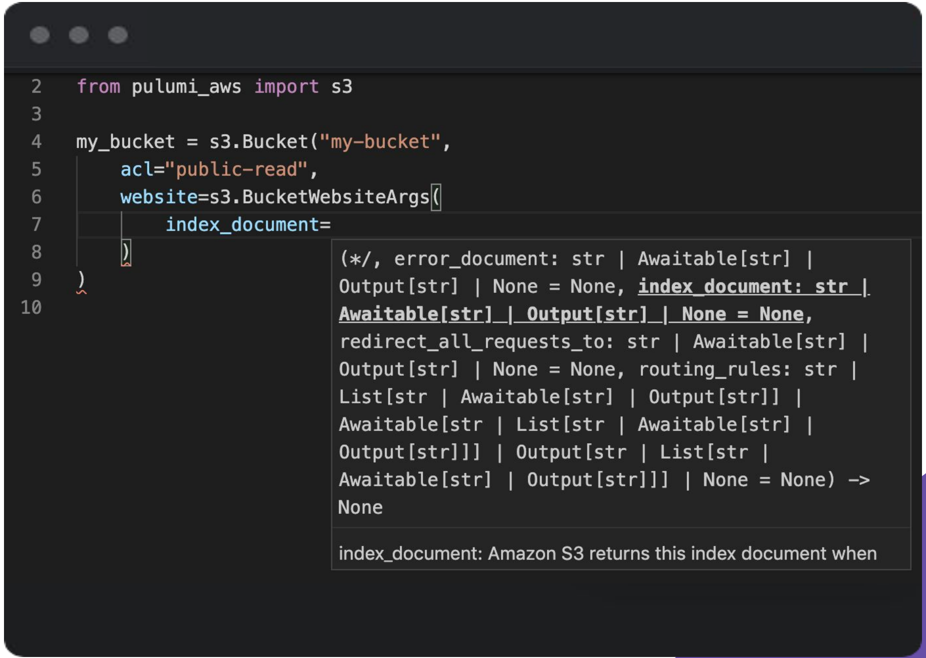
From YAML to Code

Pulumi even supports writing IaC in Pulumi YAML and converting that using `pulumi convert` to the desired programming language.

Pulumi allows you to write Infrastructure as Code in a standard programming language!

Code has a lot of advantages over *Static Configuration Languages*

- Stay with your Application Language
    - Loops, IF, ….
    - Packages/Modules, you know
- Rich IDE support
- Type checking
- Code Smells
- Create useful abstractions (Package Managers)
- Run Unit and Integration Tests
- Easy to read - very subjective □



```
from pulumi_aws import s3

my_bucket = s3.Bucket("my-bucket",
    acl="public-read",
    website=s3.BucketWebsiteArgs(
        index_document=
    )
)
```

```
(*/, error_document: str | Awaitable[str] |
Output[str] | None = None, index_document: str |
Awaitable[str] | Output[str] | None = None,
redirect_all_requests_to: str | Awaitable[str] |
Output[str] | None = None, routing_rules: str |
List[str | Awaitable[str] | Output[str]] |
Awaitable[str | List[str | Awaitable[str] |
Output[str]]] | Output[str | List[str |
Awaitable[str] | Output[str]]] | None = None) ->
None
```

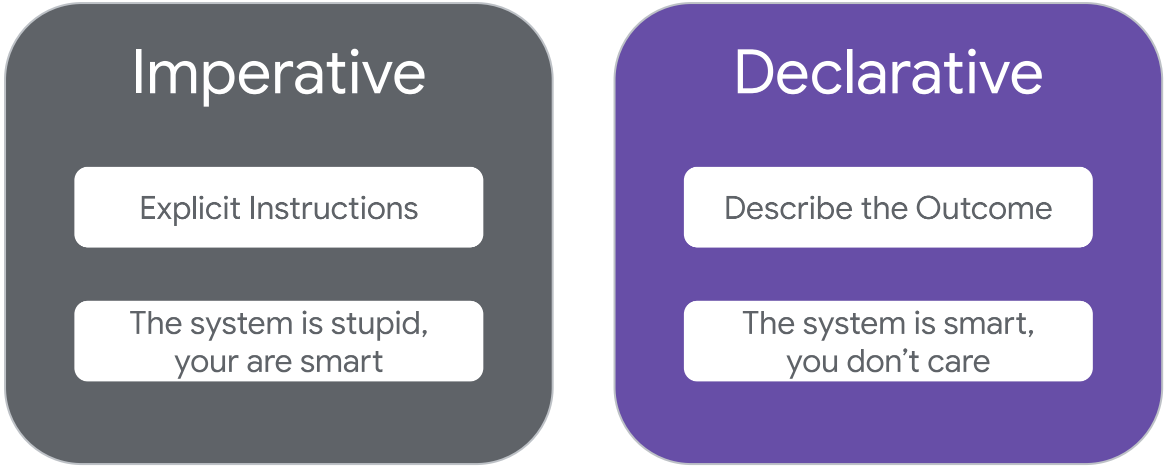
index_document: Amazon S3 returns this index document when

Pulumi

CLOUD NATIVE
COMPUTING FOUNDATION

- Free **Pulumi Open Source** - github.com/pulumi/pulumi
- vs . **Pulumi Service** (Fully-managed Cloud Engineering Platform, expensive)
- **Multi-Cloud** Capabilities & Deployments
- **Secret** Management
- **Remote-State** Handling
- Multiple Languages Supported
    - GO TS JS C# YAML
- Stack configurations for handling multiple environments

Imperative vs. Declarative



Imperative

Explicit Instructions

The system is stupid, your are smart

Declarative

Describe the Outcome

The system is smart, you don't care

Pulumi

Pulumi might use **imperative** programming languages, but you use Pulumi in a **declarative** way! You declare the resources and config and Pulumi figures out the imperative steps to reach this state.

Want to dive deeper? Check out the official Pulumi blog on Imperative vs. Declarative nature.

Pulumi vs. Terraform

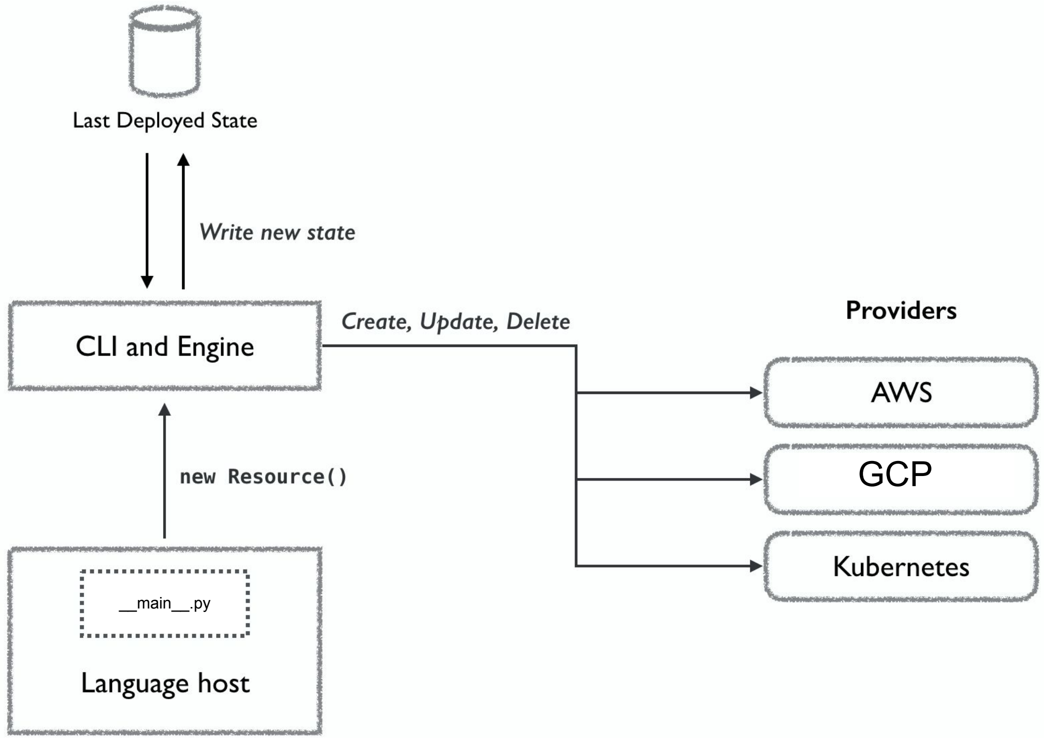
Feature	Pulumi	Terraform
Language Support	Python, TypeScript, JavaScript, Golang, C#, F#, Java, YAML, and CUE	HashiCorp Configuration Language (HCL)
Maturity	Some lack of documentation. Mid-size community.	Very mature. Large community.
Cloud Native Support	Richly typed. Includes CRDs & in-cluster operator support for GitOps delivery.	Core API typed. Generic support for CRD.
Reuse and Modularity	Flexible. Reuse functions, classes, packages, and Pulumi components.	Constrained. Can only reuse Terraform modules.
Modes of Execution	Run CLI commands or initiate commands programmatically with Automation API.	Run CLI commands or perform remote runs with SaaS offering.
Import code from other IaC	Yes. It allows to convert templates from Terraform HCL, Kubernetes YAML, Azure ARM, etc. into Pulumi programs.	No
State Management	Native support for remote State Handling.	Native support for remote State Handling.
Secret Management	Secretes can be managed remotely in Secret Manager. Secrets are encrypted in state and transit.	Difficult to prevent Secrets ending up in state file.

Pulumi Architecture

Language Host: A language executor, which is a binary, that Pulumi uses to launch the runtime for the language your program is written in.

Deployment Engine: It is responsible for computing the set of operations needed to drive the current state of your infrastructure into the desired state expressed by your program.

Resource Provider: A binary used by the deployment engine to manage a resource.



Last Deployed State

Write new state

CLI and Engine

Create, Update, Delete

Providers

AWS

GCP

Kubernetes

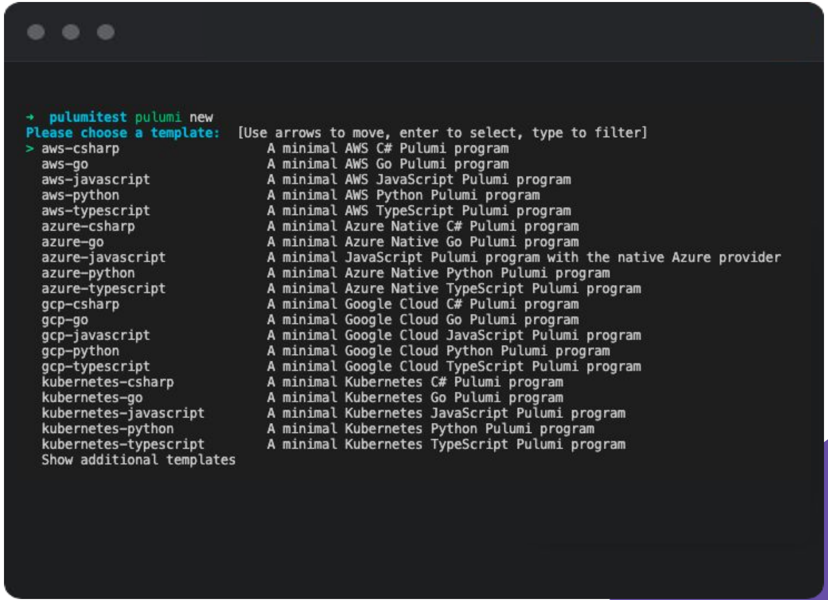
`new Resource()`

__main__.py

Language host

Language Host

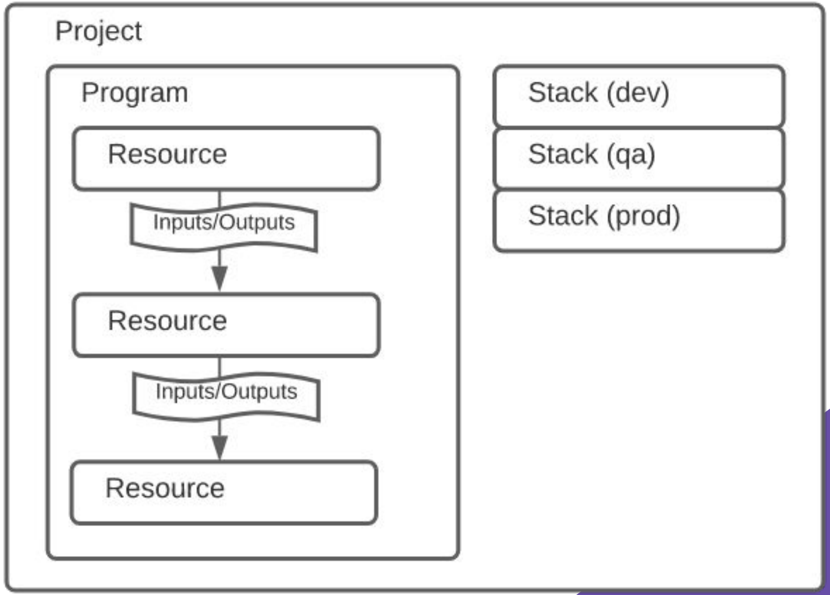
- Starting fast with **templates** ($ pulumi new)



```
➜  pulumitest pulumi new
Please choose a template:  [Use arrows to move, enter to select, type to filter]
> aws-csharp                    A minimal AWS C# Pulumi program
  aws-go                        A minimal AWS Go Pulumi program
  aws-javascript                A minimal AWS JavaScript Pulumi program
  aws-python                    A minimal AWS Python Pulumi program
  aws-typescript                A minimal AWS TypeScript Pulumi program
  azure-csharp                  A minimal Azure Native C# Pulumi program
  azure-go                      A minimal Azure Native Go Pulumi program
  azure-javascript              A minimal JavaScript Pulumi program with the native Azure provider
  azure-python                  A minimal Azure Native Python Pulumi program
  azure-typescript              A minimal Azure Native TypeScript Pulumi program
  gcp-csharp                    A minimal Google Cloud C# Pulumi program
  gcp-go                        A minimal Google Cloud Go Pulumi program
  gcp-javascript                A minimal Google Cloud JavaScript Pulumi program
  gcp-python                    A minimal Google Cloud Python Pulumi program
  gcp-typescript                A minimal Google Cloud TypeScript Pulumi program
  kubernetes-csharp             A minimal Kubernetes C# Pulumi program
  kubernetes-go                 A minimal Kubernetes Go Pulumi program
  kubernetes-javascript         A minimal Kubernetes JavaScript Pulumi program
  kubernetes-python             A minimal Kubernetes Python Pulumi program
  kubernetes-typescript         A minimal Kubernetes TypeScript Pulumi program
  Show additional templates
```

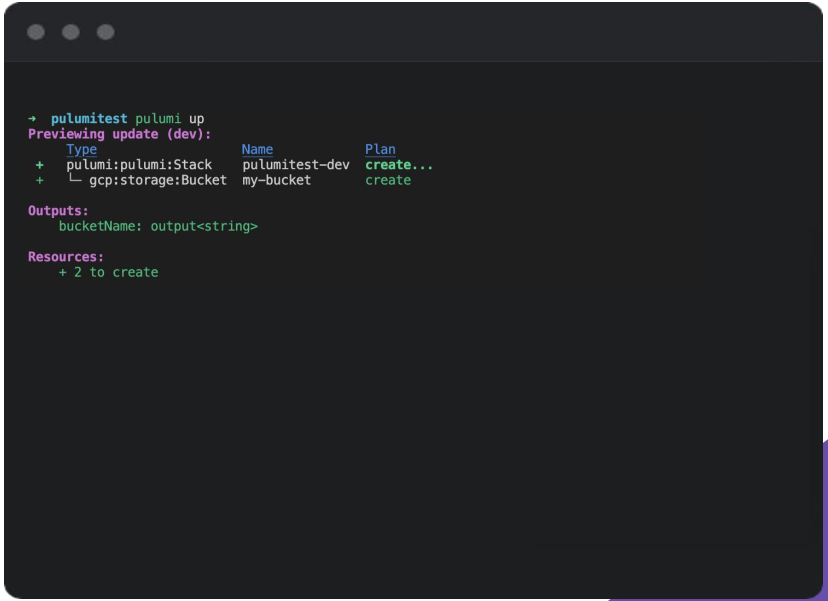
Language Host

- Starting fast with **templates** ($ pulumi new)
  - Creates a project and boilerplate program
- **Configurations** for different **stacks** are
  - Isolated and independently configurable instance of a Pulumi program
  - Configuration variables and secrets in from of **tags**
  - Capable of referencing from other stacks - very powerful
- **Code** itself - __main__.py, …
  - create Stack outputs
  - … and can import other stack's output!



Deployment Engine

- Determine changes for required state
  - Create, Update, Delete Resources via a Provider



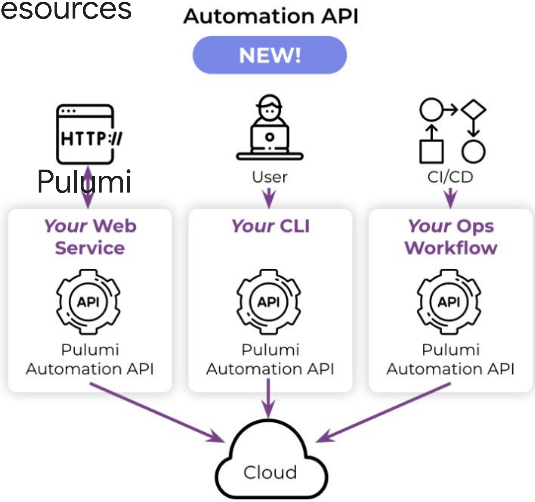
```
➜  pulumitest pulumi up
Previewing update (dev):
     Type                    Name            Plan
 +   pulumi:pulumi:Stack     pulumitest-dev  create...
 +   └─ gcp:storage:Bucket   my-bucket       create

Outputs:
    bucketName: output<string>

Resources:
    + 2 to create
```

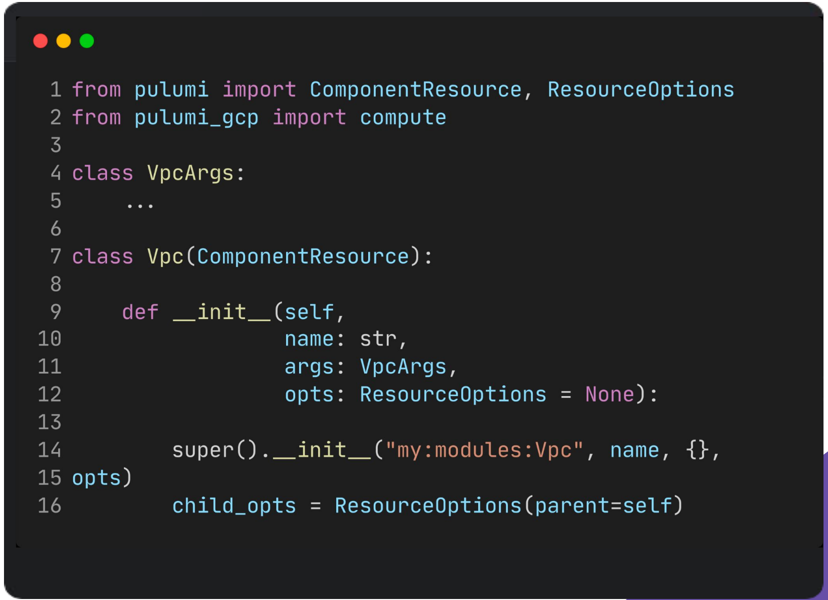
Deployment Engine

- Determine changes for required state
  - Create, Update, Delete Resources via a Provider
- **Automation API**
  - Programmatic interface for running Pulumi programs without the Pulumi CLI
  - Run Pulumi and your IaC as executable



Component Resources

- **Abstraction:** A set of logical grouping of resources and Config
- The implicit `***pulumi:pulumi:Stack***` resource is itself a component resource that contains all top-level resources in a program
- **A few examples of Component Resources:**
  - A ***VPC*** that automatically comes with built-in best practices
  - A ***KubernetesCluster*** that can create EKS, AKS, and GKE clusters, depending on the target
- ***Want to create a new Component Resource?***
  - *Extend from the* ***ComponentResource*** *class*



```
from pulumi import ComponentResource, ResourceOptions
from pulumi_gcp import compute

class VpcArgs:
    ...

class Vpc(ComponentResource):

    def __init__(self,
                 name: str,
                 args: VpcArgs,
                 opts: ResourceOptions = None):

        super().__init__("my:modules:Vpc", name, {},
opts)
        child_opts = ResourceOptions(parent=self)
```

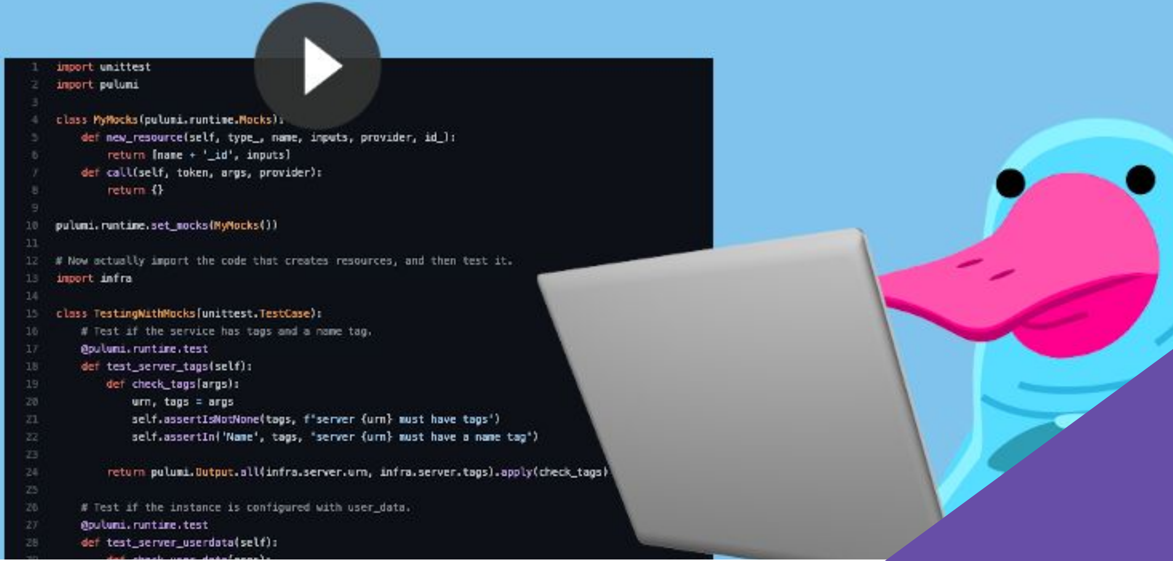
Pulumi Convertors

Pulumi converters allow you to convert ARM (Azure Resource Manager), CloudFormation, Kubernetes Custom Resources, Kubernetes YAML, and Terraform to Pulumi.

- ARM to Pulumi: This conversion tool will do the magic of translating your ARM templates into modern code using Pulumi.
- CloudFormation to Pulumi: This conversion tool will do the magic of translating your CloudFormation templates into TypeScript/JavaScript, Python, Golang, and C# using Pulumi.
- Kubernetes CustomResources to Pulumi: CustomResources in Kubernetes allow users to extend the API with their types. These types are defined using CustomResourceDefinitions (CRDs), which include an OpenAPI schema. The new crd2pulumi tool takes the pain out of managing CustomResources by generating types in the Pulumi-supported language of your choice!
- Kubernetes YAML to Pulumi: This conversion tool will do the magic of translating your Kubernetes YAML into modern code using Pulumi.
- Terraform to Pulumi (tf2pulumi): This conversion tool will do the magic of translating your HCL into modern code using Pulumi.
- Pulumi even supports writing IaC in Pulumi YAML and converting that using `pulumi convert` to the desired programming language.

Pulumi

Demo Time



```python
import unittest
import pulumi

class MyMocks(pulumi.runtime.Mocks):
    def new_resource(self, type_, name, inputs, provider, id_):
        return [name + '_id', inputs]
    def call(self, token, args, provider):
        return {}

pulumi.runtime.set_mocks(MyMocks())

# Now actually import the code that creates resources, and then test it.
import infra

class TestingWithMocks(unittest.TestCase):
    # Test if the service has tags and a name tag.
    @pulumi.runtime.test
    def test_server_tags(self):
        def check_tags(args):
            urn, tags = args
            self.assertIsNotNone(tags, f'server {urn} must have tags')
            self.assertIn('Name', tags, 'server {urn} must have a name tag')

        return pulumi.Output.all(infra.server.urn, infra.server.tags).apply(check_tags)

    # Test if the instance is configured with user_data.
    @pulumi.runtime.test
    def test_server_userdata(self):
```

GeoPython 2023

Basel Switzerland March 06-08

👋 Thanks Everyone !

Deck: tinyurl.com/geopython-pulumi-2023

```
title: Code-Centric Infrastructure
as Code (IaC) using Pulumi with
Python
speaker: Anmol Krishan Sachdeva
job: Hybrid Cloud Architect
company: Google
handle: @greatdevaks
```