Delivering Cloud Native Infrastructure as Code

Enabling the future of cloud engineering with Pulumi

pulumi

Executive Summary

To a first approximation, all developers are cloud developers, all applications are cloud native, and all operations are cloud-first. Yet, there is a lack of a consistent approach to delivering cloud native applications and infrastructure. The tools and processes differ by technology generation, and even by cloud vendor, and so deny the full potential of cloud native application delivery.

In this paper, we make the case for a consistent programming model for the cloud and examine:

- How the cloud has already evolved three times as it increasingly moves toward stateless compute to deliver on the opportunities afforded by unprecedented economies of scope and scale.
- How stateless compute has shifted infrastructure management concerns from 'at rest' to 'in motion', and increasingly moved these concerns up the stack to development.
- How the growth of DSL-based tools has lead to complexity for DevOps teams, failed to deliver on the promised collaboration between development and operations functions, and does not satisfy the need for increasing delivery speed in the cloud.

We explore how Pulumi is designed to deliver on cloud native infrastructure as code requirements with a consistent programming model, providing a development platform and control plane to meet the demands of high-velocity application development in the cloud.

{ *To a first approximation, all developers are cloud developers, all applications are cloud native, and all operations are cloud-first.*

The Challenges of Cloud Development and DevOps

The cloud is no longer new, and most IT teams — from development to operations — will have cloud as some portion of their portfolio, and likely a long-term strategy or posture on cloud migration. But precisely how has the cloud changed the tools, and concerns for software delivery?

The 3 Evolutions of Cloud

Enterprise software has undergone a slow shift from containerless servers to serverless containers. In its relatively short lifespan, the cloud has already undergone — or is undergoing — three distinct evolutions. Each of these evolutions unlocked further economies of scope and scale.

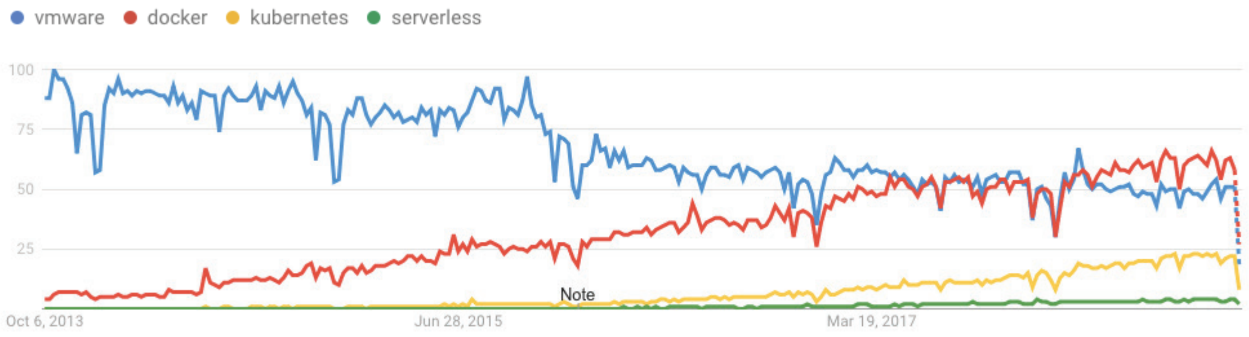
- **Virtual Machines.** The initial wave of cloud computing offered 'lift and shift' migration strategies and flipped infrastructure planning from a capex to opex activity.
- **Containers (and Kubernetes).** The current wave of cloud computing offered improvements over VMs well-suited to cloud native approaches: infrastructure abstraction, OS isolation, and a model for highly distributed applications.
- **Serverless.** The incoming wave of cloud computing offers the current ultimate role for the cloud: a stateless mesh of on-demand, infinitely scalable services, with options for arbitrary code execution.

In doing so, cloud computing has been continuously driving towards event-driven stateless computing. The current state-of-the-art 'serverless' computing can be said to be entirely stateless, with the consumer of those services paying for state when required, in other words, when the functions are executed.

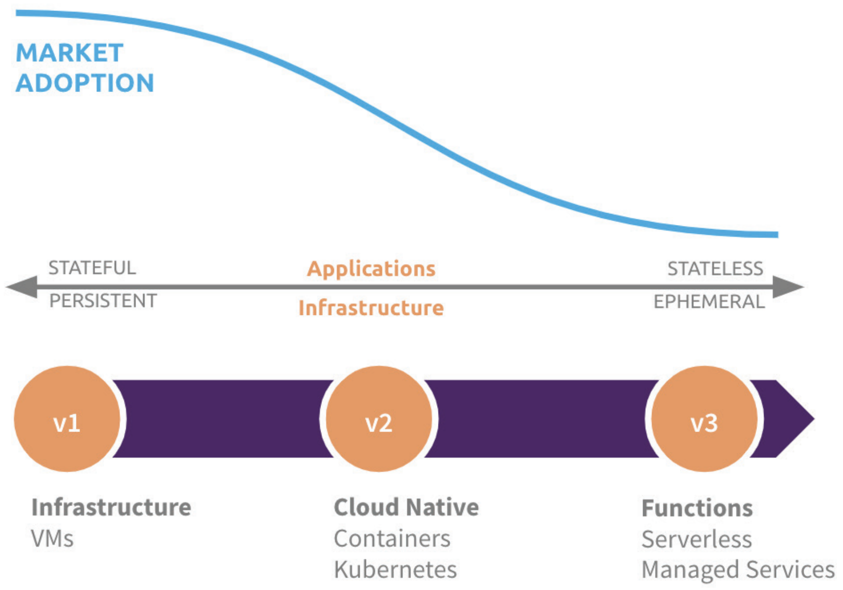
Enterprise software has undergone a slow shift from containerless servers to serverless containers.

As usual, the future is unevenly distributed. A quick look at Google Trends over five years provides a reasonably clear picture of how embedded these evolutions are in the market at large. In this view, we use the presumed category incumbents to represent the market.

INTEREST OVER TIME, 5 Years to Current Date



We can redraw this picture as an adoption curve across these evolutions approximately as follows:



Each of these evolutions has opportunity, but also embedded cost: cost to switch, cost of skills, and cost of workflows and tools. Each also carries risk through isolated stovepipes: limited numbers of experts to attend to a given paradigm.

To remediate these costs and risks, enterprises need tools that can work across these paradigms in a consistent way, and also remain consistent as the next inevitable evolution occurs.

The Shift from Infrastructure at Rest, to Infrastructure in Motion

If infrastructure isn't precisely immutable, it's certainly increasingly ephemeral. Docker, and container technology generally, champions immutable infrastructure, but as already described, not everything cloud native is immutable, nor is it desired to be immutable. However, the nature of the available endpoints in the form of managed services, or code execution environments such as serverless environments, or Kubernetes clusters, means that developers are 'choosing the infrastructure' for their apps by design.

Making infrastructure choices inside application development increases the ephemerality of the infrastructure: application code refactoring may result in significant changes to the infrastructure (or more specifically, the cloud resources necessary for execution of the code). Additionally, the use of managed services typically means a constrained and/or well-known set of configurations to apply, vs an arbitrary custom deployment.

If infrastructure isn't precisely immutable, it's certainly increasingly ephemeral.

This is a huge shift from 'infrastructure at rest' where configuration management, and convergence-based IaC tools were the preferred way of managing a typical lifecycle of provisioning, through long-lasting configuration tasks such as patch management, compliance, and drift correction. Instead, cloud native development practices mean 'infrastructure in motion', where provisioning could be tied to individual commits of application code, and the lifespan of the infrastructure (whether immutable or mutable) does not warrant continued configuration management.

This ephemerality is often characterized as 'developers moving down the stack' (becoming responsible for infrastructure) and 'operations moving up the stack' (becoming responsible for code), but the reality is the overall shift is towards increasingly codified provisioning of short-lived infrastructure.

To support this shift, enterprises need tools that can connect application code and infrastructure code in way that is logical, expressive, and familiar to development and operations teams where application code increasingly defines the infrastructure resource requirements for a cloud application.

The Trouble with DSLs

YAML Ain't A Programming Language Either. The reality is that DSLs certainly become overstretched, and overused by practitioners for a number of reasons: 1) the initial success of a 'DSL' means it naturally gets used outside of intended domain, 2) the domain itself extends beyond its initial boundaries and 3) it's often easy to build an initial solution in DSL which can become ossified. Inevitably then, DSLs are typically neither domain-specific, nor languages.

In the case of IaC, one of the most prevalent DSLs is YAML. YAML was not intended as a programming language and lacks the features and tooling necessary to make it productive. Worse, it encourages poor programming practice: for instance, repetition of huge chunks of code owing the lack of subroutine support, or abstraction mechanics.

YAML Ain't A Programming Language Either.

YAML, for instance, should be considered as bytecode: intended to be an output format of some other program. Instead, development and devops teams wrestle with ever growing, and increasingly fragile, lines of YAML code, without the support of sophisticated IDEs and similar tooling support to make them productive.

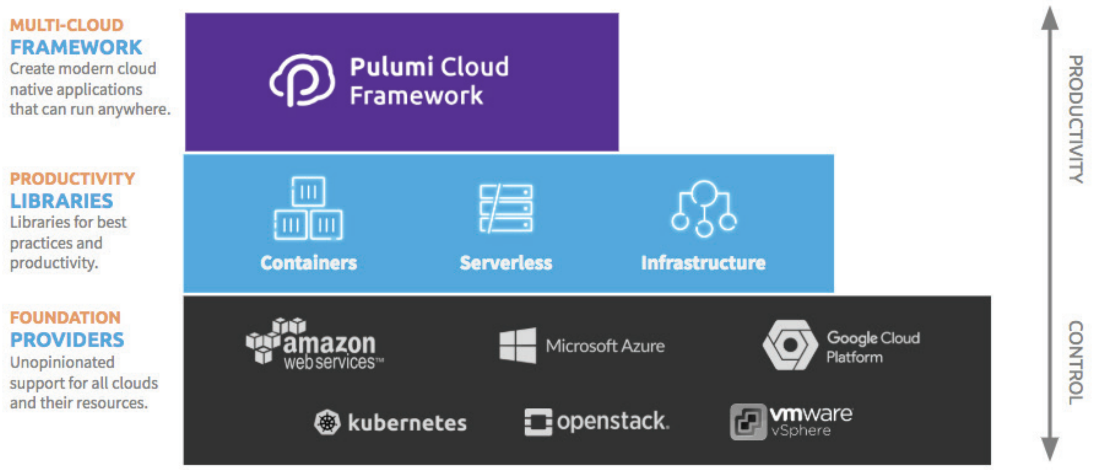
DevOps teams have taken it upon themselves to become those DSL experts, but are increasingly overwhelmed by the number of DSLs needed for cloud-based DevOps. Development teams on the other hand, simply reject non-code-based DSLs and spend their time working on the things that matter to them: the app. Those teams that embraced DevOps as means of improving software delivery speed and quality through collaboration and shared practices find themselves once again isolated.

To remediate the strain of DSLs on development and devops teams, enterprises need tools that provide consistent, consolidated experiences, and provide the mature tooling that development teams expect in other aspects of their work.

Pulumi Provides a Cloud Native Programming Model

Using real languages changes everything. The Pulumi Cloud Development Platform is a combination of tools, libraries, runtime, and service that delivers a consistent development and operational control plane for cloud native infrastructure.

At the center of Pulumi is an open source cloud object model, coupled with an evaluation runtime that understands how to take programs written in any language, understand the cloud resources necessary to execute them, and then plan and manage those resources in a robust way. This cloud runtime and object model is inherently language- and cloud-neutral, enabling Pulumi to support many languages and clouds rapidly.



Pulumi aims to provide a solution to the challenges of cloud application development and delivery by providing a consistent programming model for cloud native development:

- **Consistent across cloud technology paradigms.** Providing a single toolset for infrastructure, managed services, containers, and serverless environments on any cloud infrastructure.
- **Productive, expressive use of real languages.** Providing a common runtime that can be bound to existing languages, taking advantage of the idioms of that language: from abstraction, through to packaging.
- **Continuous delivery for cloud native infrastructure.** Providing a common structure to connect application code and infrastructure code to ensure a rapid inner development loop, and a well-managed outer operational loop.

Consistency and power across cloud technology paradigms.

The fragmentation of existing tools and DSLs is consolidated within the Pulumi framework which offers:

Multi-Language Runtime. The Pulumi runtime was architected to support many languages, and to do so in an idiomatic way for all aspects of a target language: style, syntax, packages, etc. It currently supports JavaScript, TypeScript, Python, and Go and is fully open source to accept contributions for alternative languages.

Cloud Object Model. Pulumi's underlying cloud object model offers a rich view into how cloud programs are constructed. The resulting objects form a Directed Acyclic Graph (DAG) using dependencies from program that the system can analyze and understand to deliver insights for sophisticated static analysis and visualizations.

Multi-Technology Scope. Pulumi's programming model extends across infrastructure, managed services, containers, container management, and serverless technology areas. The full APIs of those servers are exposed and can be targeted by a chosen language. Targeted technologies may be combined to unlock the scenarios that enterprises need: e.g. connecting data services to Kubernetes clusters, or combining serverless functions with container tasks.

Multi-Cloud Scope. The Pulumi cloud object model is a powerful foundation that can support any cloud provider. This delivers a unified programming model, tools, and control plane for managing cloud software anywhere. There's no need to learn a variety of DSLs and tools just to get a cloud native application into production. Pulumi supports the major cloud vendors (Amazon Web Services, Microsoft Azure, Google Cloud Platform), Kubernetes (in any environment), along with OpenStack and VMWare vSphere.

Pulumi makes multi-cloud, multi-technology targeting a reality with a consistent programming model in real languages.

{ *Using real languages changes everything.*

Productive, expressive use of real languages.

As the Pulumi runtime can support many languages, development and devops teams gain the natural productivity benefits of those languages which are often missing from DSLs and associated tooling. Advantages include:

Familiarity and expressiveness. Using already understood languages has clear productivity, communication and collaboration benefits and negates the need to learn new bespoke DSLs or YAML-based templating languages. Developers and DevOps teams can work using a common language and code base rather than attempting a hand-off through a DSL-enforced separation of concerns, which improves collaboration, and reduces the fragility and risk in a system.

Real languages also means gaining access to powerful - but obvious - benefits: capture references to variables such as constants, configuration settings or encrypted secrets, or even references to other resources so that all resources can be connected in an expressive and logical manner to achieve the desired output.

```
import * as aws from “@pulumi/aws”;
import * as serverless from “@pulumi/aws-serverless”;
let topic = new aws.sns.Topic(“topic”);
serverless.cloudwatch.onEvent(“hourly”, “rate(60 minutes)”, event => {
        const sns = new (await import “aws-sdk”).SNS();
        return sns.publish({
        Message: JSON.stringify({ event: event }),
        TopicArn: topic.id.get(),
        }).promise();
});
```

Abstraction and reuse. Real development languages offer abstraction, and as a result, reuse of code. Pulumi supports the appropriate package managers for the chosen development language (e.g. NPM for JS/TS, PyPi for Python). This allows the elimination of significant LoC from a typical DSL-based configuration, and removes the practice of copy-and-paste development with its inherent drift and error replication risks.

This example code shows the refactoring of a typical AWS Best Practice for setting up a VPC, and how it can be reused and instantiated as needed in new Pulumi programs.

```
import * as awsinfra from “@pulumi/aws-infra”;
let network = new awsinfra.Network(`${prefix}-net`, {
        numberOfAvailabilityZones: 3, // Create subnets in many AZs
        usePrivateSubnets: true,    // Run inside private per-AZ subnets
});
```

Tooling and workflows. By using real languages, development and devops teams instantly gain access to IDEs, refactoring, testing, static analysis and linters, and so much more. This both improves the productivity and quality of team efforts, and throws into harsh relief the lack of tooling for DSL-based approaches, which is often hard to debug and remediate.

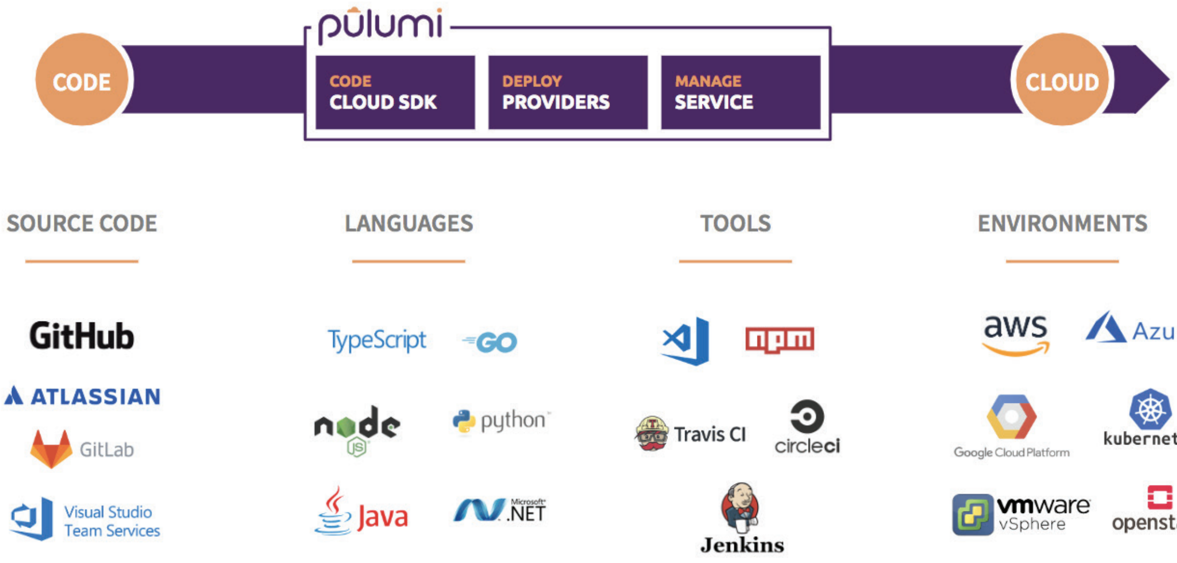
Pulumi brings the existing power of real programming languages to cloud native infrastructure and application delivery for huge productivity and collaboration gains.

{ *Pulumi brings the existing power of real programming languages to cloud native infrastructure and application delivery for huge productivity and collaboration gains.*

Continuous delivery for cloud native infrastructure.

Because infrastructure is now linked to application code, and because of the ephemeral nature of the infrastructure, the 'integration' aspect of CI/CD becomes less interesting. What matters more is the 'delivery' aspect: improving both 'inner loop' development and 'outer loop' operations.

Previously there was very limited tooling at the very point of collaboration needed by development and devops teams. Pulumi connects those teams and improves the required workflows.



Delivering Cloud 'Stacks'. A core concept in Pulumi is the idea of a "stack." A stack is an isolated instance of a cloud program whose resources and configuration are distinct from all other stacks. A team might have a stack each for production, staging, and testing, or perhaps for each single-tenanted environment. Pulumi's CLI makes it trivial to spin up and tear down lots of stacks. This opens up workflows that might not have previously even attempted, such as each developer having her own stack, spinning up (and tearing down) a fresh stack to test out each Pull Request, or even splitting tiers of your service into many stacks that are linked together - all of which is applicable and useful in cloud application deliver scenarios.

Enabling a unified Cloud Native Infrastructure CD Pipeline. As a Pulumi stack can contain references to any aspect of a cloud environment then it provides a unified pipeline for continuous delivery. For instance, Pulumi can setup and configure managed infrastructure, build and publish containers into orchestration systems, and connect serverless functions all in the same code and stack. The Pulumi runtime can then be invoked to build, update, and teardown this infrastructure at will.

Pulumi rewrites the rulebook on CI/CD by connecting application code and infrastructure code at the code base, unlocking new workflows to deliver cloud native infrastructure as code.

Use Cases

Pulumi provides a consistent programming model across the cloud, from VMs through containers and Kubernetes, to serverless and managed services.

Infrastructure. Managed cloud services and infrastructure, continuously deployed and configured in a robust and compliant manner.

```
// Create a simple web server
const aws = require("@pulumi/aws");
let size = "t2.micro";
let ami = "ami-7172b611"

let server = new aws.ec2.Instance("web-
server-www", {
    tags: { "Name":"web-server-www" },
    instanceType: size,
    securityGroups: [ group.name ],
    ami: ami,
    userData: userData
});

exports.publicIp = server.publicIp;
exports.publicHostName = server.publicDns;
```

Kubernetes. Target on-premises or cloud-based Kubernetes services to provision clusters, and create, deploy, and manage apps.

```
// Deploy 3 replicas of an nginx pod
import * as k8s from "@pulumi/kubernetes";
function deploy(name, replicas, pod) {
    return new k8s.apps.v1beta1.Deployment(name, {
        spec: {
            selector: { matchLabels: pod.metadata
labels },
            replicas: replicas,
            template: pod
        }
    });
}
const nginxServer = deploy("nginx", 3, {
    metadata: { labels: { app: "nginx" } },
    spec: {
        containers: [{ name: "nginx",
                image: "nginx:1.15-alpine" }]
    }
});
```

Serverless. Deploy and scale websites easily, handle event-streaming, and processing with multi-cloud microservices.

```
// Create a serverless REST API
import * as cloud from "@pulumi/cloud";
let app = new cloud.API("my-app");
app.static("/", "www");

app.get("/hello", (req, res) =>
       res.json({ hello: "World!" }));

export let url = app.publish().url;
```

Containers. Deploy container-based apps into any cloud native infrastructure, from VMs to Kubernetes, to custom orchestrators.

```
// Deploy a customer nginx container
import * as cloud from "@pulumi/cloud";
let nginx = new cloud.Service("nginx", {
    build: ".",
    ports: [{ port: 80 }],
    replicas: 2,
});

export let url = nginx.defaultEndpoint;
```

Case Study:

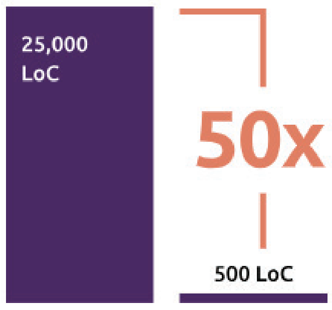
Learning Machine migrates from CloudFormation to Pulumi

Learning Machine, a successful blockchain-based SaaS company, faced two challenges with their cloud native infrastructure:

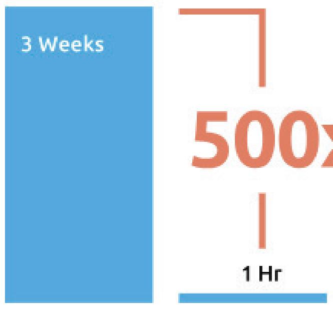
1. Increasing skills gaps between development and devops teams began to create silos, and risk to their code base.
2. Increasing need to more rapidly provision their service owing to their expanding roster of new customers.

Learning Machine suffered from a loss of productivity and an inability to meet business demands, as the development team did not understand the systems, tools, or architecture for service provisioning.

By using Pulumi, Learning Machine were able to reduce 25,000 LoC of ad-hoc scripts to 500 LoC of JavaScript that could be understood across all teams, enabling the development team to take accountability for service delivery to meet business needs. Additionally, moving to Pulumi removes lock-in to a specific cloud, and has enabled Learning Machine to begin work on their on-premises private cloud service.



25,000 Lines of CloudFormation reduced to 500 Lines of JavaScript



New customer provisioning time reduced from 3 weeks to 1 hour

"Pulumi has given our team the tools and framework to achieve a unified development and DevOps model, boosting productivity and taking our business to any cloud environment that our customers need. We retired 25,000 lines of complex code that few team members understood and replaced it with 100s of lines in a real programming language."

Kim Hamilton, CTO, Learning Machine

Conclusion

The rapid pace of evolution of the cloud, combined with the shift to ephemeral infrastructure, and the connection of application code and infrastructure code, demands a different view of cloud development and devops.

Pulumi provides a platform to deliver cloud native infrastructure as code for any technology, any cloud, and any language:

- Making multi-cloud, multi-technology targeting a reality with a consistent programming model in real languages.
- Using the existing power of real programming languages to cloud native infrastructure and application delivery for huge productivity and collaboration gains.
- Rewriting the rulebook on CI/CD by connecting application code and infrastructure code at the code base, unlocking new workflows to deliver cloud native infrastructure as code.

Cloud Native development represents a step change in opportunity, and capabilities, for development and devops teams. Pulumi brings together those teams to ensure their success through huge productivity and quality gains with a Cloud Native Development Platform designed for every cloud evolution.

{ *The rapid pace of evolution of the cloud, combined with the shift to ephemeral infrastructure, and the connection of application code and infrastructure code, demands a different view of cloud development and devops.*

About Pulumi

Pulumi provides a Cloud Native Development Platform that enables teams to get their serverless, containers, infrastructure and Kubernetes code to the cloud quickly and simply. Pulumi provides an SDK to define, deploy and manage cloud services using familiar languages (JavaScript, TypeScript, Python, Go), with clear collaboration for teams and enterprises to deliver cloud native, multi-cloud capable, applications and solutions. For more, visit http://pulumi.com