Multi-Language Infrastructure as Code

The Cloud is the New Operating System: Let's Program It

Joe Duffy
@funcOfJoe
Pulumi Founder and CEO

6/26/19 - QCon NYC

Why Infrastructure as Code?
Standing on the Shoulders of Giants

Why Infrastructure as Code?

Standing on the Shoulders of Giants

My Background

All developer tools:

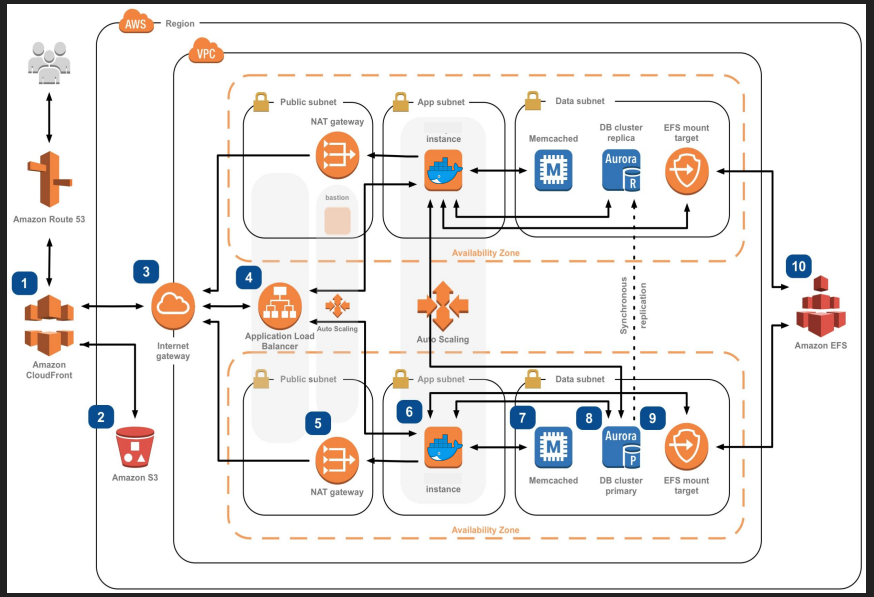
- Early engineer on .NET and C#.
- Created team that built Task/Async in .NET.
- Architect for safe, distributed operating system.
- Led languages groups (C++/C#/F#/etc), IDE support, static analysis.
- Initiated effort to open source .NET and take to Linux/Mac.

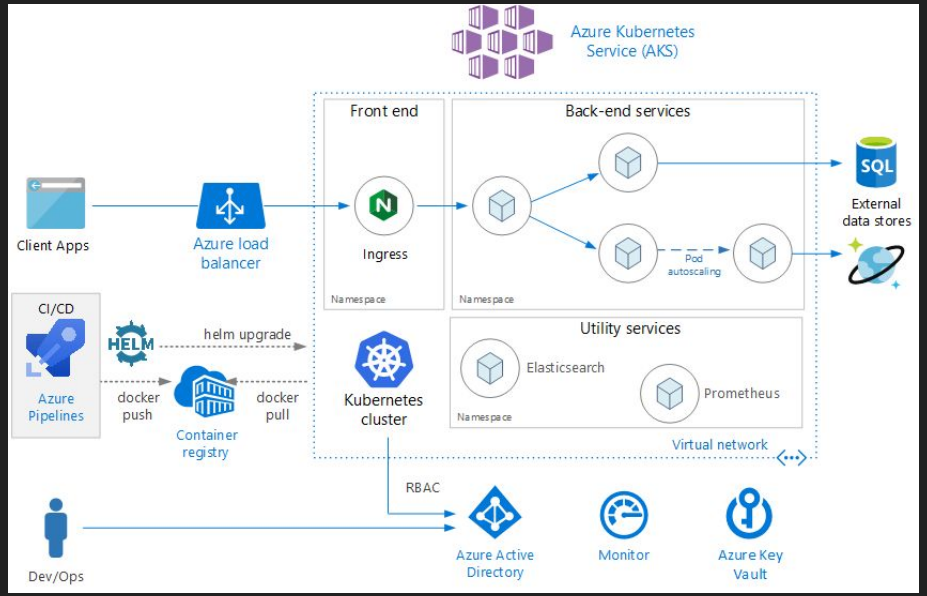
Came to the cloud from a different perspective.

I didn’t know I’d be doing infrastructure as code, until we started doing it ...

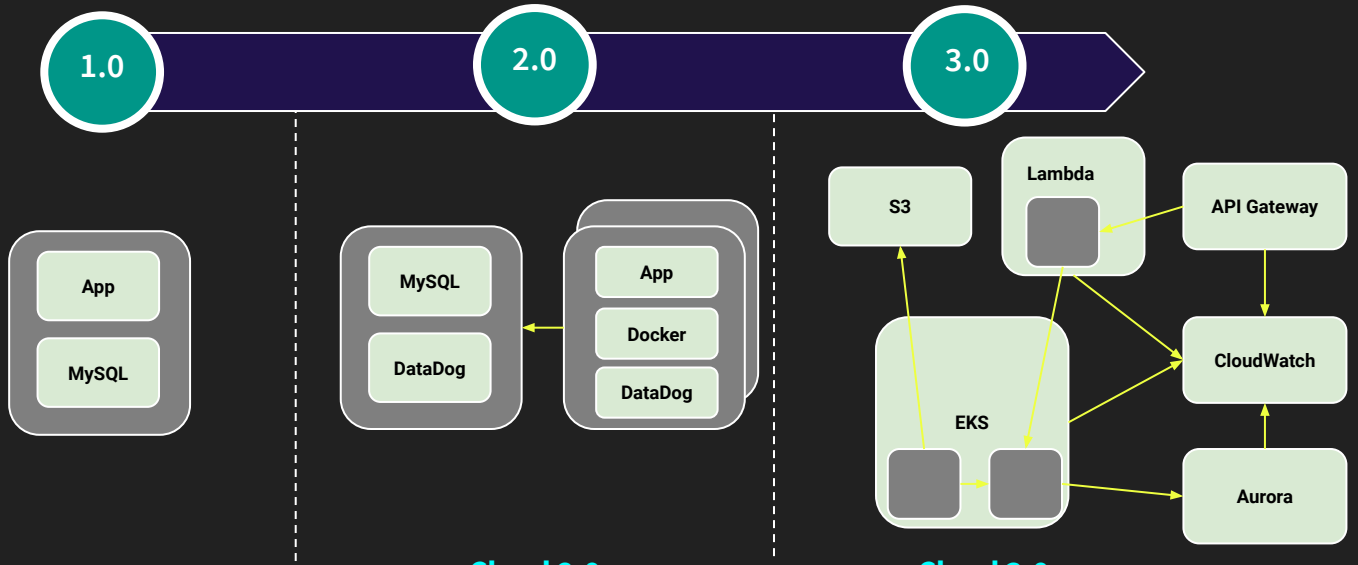
All developers are (or will become) cloud developers.

I’m a Developer -- Why Infrastructure?





Modern Cloud Architectures



Cloud 1.0
2000-2009
- Fixed VMs
- N-Tier Apps
- Private Cloud

Cloud 2.0
2010-2019
- Dynamic VMs
- Hosted DBs
- Hybrid Cloud

Cloud 3.0
2019+
- Serverless
- Containers
- Public Cloud

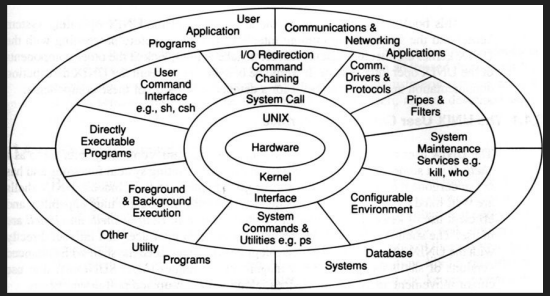
The cloud is no longer an afterthought.

The Cloud Operating System

What is an operating system anyway?

- Provides HW resources to our applications (CPU, network, memory, disk).
- Manages competing demands through scheduling.
- Secures access to resources.
- Offers primitives, and application models, that developers and IT admins use to get things done without meddling with hardware.

```
s/operating system/cloud/g
```



The Cloud *is* the Operating System

	Traditional OS	Cloud OS
Granularity	One Machine	Fleets of Machines
Master	Kernel	Control Plane
“Perimeter”	NIC/Firewall	Virtual Private Cloud
Security	ACLs, Users/Groups/Roles	IAM (Users/Groups/Roles)
Scheduling	Processes and Threads	VMs, Containers, Functions
Storage	Filesystem, Registry	Block Store, Objects, Databases
Packaging	Executables, Shared Libraries	Images (VMs, Containers, Functions)
Debugging	In-Memory/Interactive	Logging/Postmortem

Managing Infrastructure

From **kernel objects** to **infrastructure resources**:

- Networks, security roles
- Virtual machines, Kubernetes clusters, private Docker repositories
- Databases, object stores, AI services

How do we manage the lifecycle for these infrastructure resources?

	Repeatable?	Reviewable?	Reliable?	Versionable?
Point and click	😟	😟	😟	😟
Scripts	😕	😕	😕	😕
Infrastructure as Code	😁	😁	😁	😁

Infrastructure as Code lets you declare cloud resources — clusters, compute, databases, hosted services — that your application needs, using code.

Declarative Infrastructure as Code takes those declarations, compares the “goal” to the “current” state of your cloud, and rectifies the difference.

From Scripting...

```
$ SECGROUP_ID=$(
    aws ec2 --region us-east-1 \
        create-security-group \
            --group-name="web" \
            --description="Web")


$ aws ec2 --region us-east-1 \
    authorize-security-group-ingress \
        --group-id=$(SECGROUP_ID) \
        --protocol="tcp" \
        --port=80 \
        --cidr="0.0.0.0/0"


$ aws ec2 --region us-east-1 \
    run-instances \
        --ami-id=ami-25488752 \
        --instance-type=t2.micro \
        --security-group-ids=${SECGROUP_ID} \
        --user-data=template/user_data.sh \
        --tag-specifications=\
            "[{Key=Name,Value=hello-world-web}]"
```

To Infrastructure as Code...

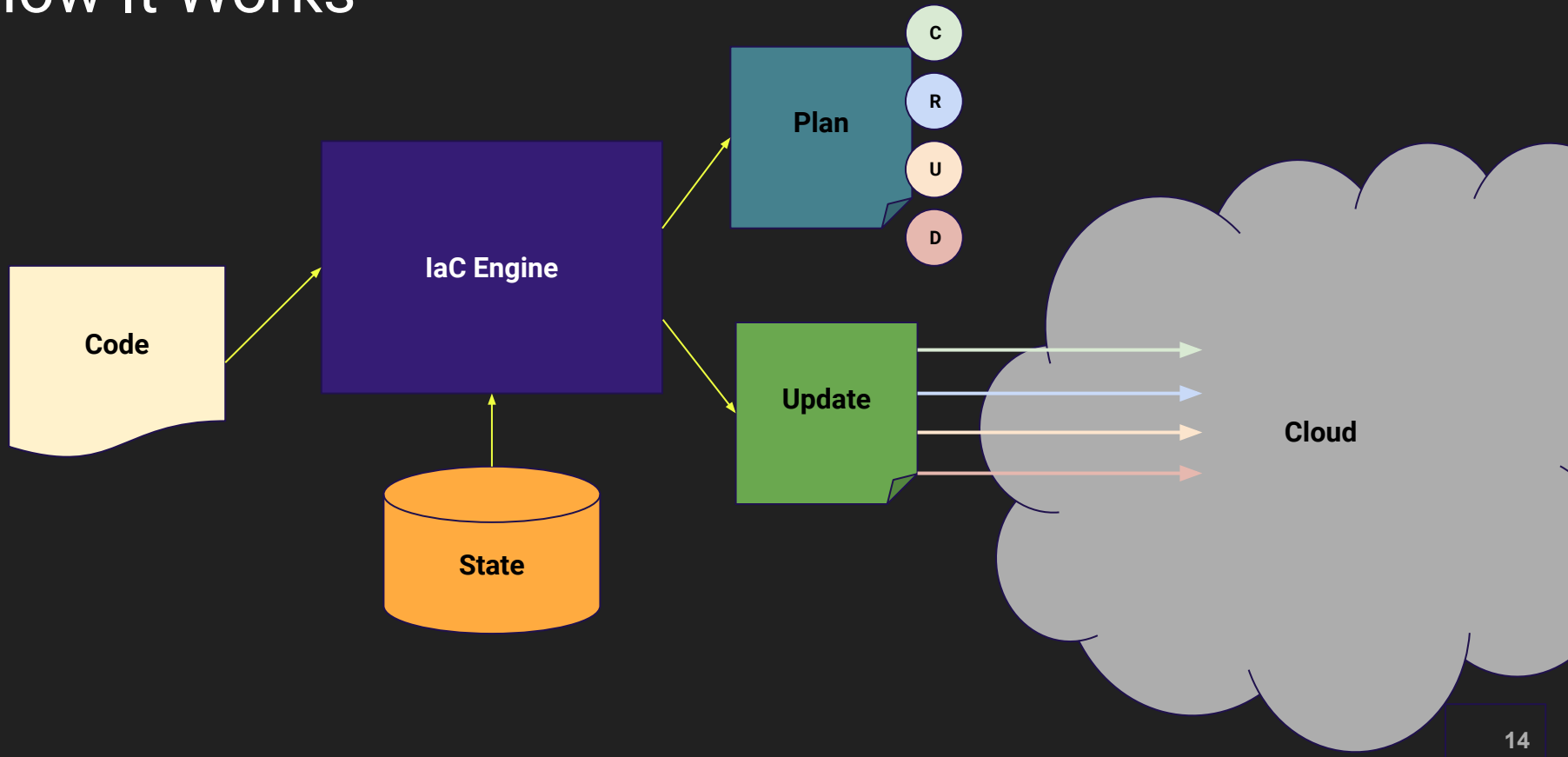
```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web" {
  ami                    = "ami-25488752"
  instance_type          = "t2.micro"
  vpc_security_group_ids = ["${aws_security_group.web.id}"]
  user_data              = "${file("template/user_data.sh")}"

  tags {
    Name = "hello-world-web"
  }
}

resource "aws_security_group" "web" {
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

How It Works



Days 1, 2, and Beyond

Day 1: standing up new infrastructure

- For a new project
- For taking a prototype into production

Day 2+: evolving existing infrastructure

- Continuously deploying application updates
- Upgrading to new versions of things (e.g., Kubernetes 1.14 to 1.15)
- Evolving topology as needs change (e.g., adding a new microservice, scaling out an existing service, leveraging new data services, adopting new best practices)
- For a new environment for an existing product (e.g., dev/stage/prod1/prod2)

Just One Problem...

Infrastructure as code is often **not code**! 😢

- YAML, domain specific languages (DSLs), ...
- Breaks down at scale -- begets templates and YAML mungers.

We're missing a lot of things we love about code!

- Abstraction and reuse: functions, classes
- Expressive constructs: loops, list comprehensions
- Great tooling: IDEs, refactoring, linters, static analysis
- Most of all, productivity!

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "wordpress.fullname" . }}
  labels:
    app: "{{ template "wordpress.fullname" . }}
    chart: "{{ template "wordpress.chart" . }}"
    release: {{ .Release.Name | quote }}
    heritage: {{ .Release.Service | quote }}
spec:
  selector:
    matchLabels:
      app: "{{ template "wordpress.fullname" .
      release: {{ .Release.Name | quote }}
  {{- if .Values.updateStrategy }}
  strategy: {{ toYaml .Values.updateStrategy |
  {{- end }}
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: "{{ template "wordpress.fullname"
        chart: "{{ template "wordpress.chart"
        release: {{ .Release.Name | quote }}
{{- if or .Values.podAnnotations .Values.metric
      annotations:
{{- if .Values.podAnnotations }}
{{ toYaml .Values.podAnnotations | indent 8 }}
{{- end }}
{{- if .Values.metrics.podAnnotations }}
{{ toYaml .Values.metrics.podAnnotations | inde
{{- end }}
{{- end }}
    spec:
      {{- if .Values.schedulerName }}
      schedulerName: "{{ .Values.schedulerName
      {{- end }}
{{- include "wordpress.imagePullSecrets" . | in
```

```
      hostAliases:
       - ip: "127.0.0.1"
         hostnames:
         - "status.localhost"
       containers:
       - name: wordpress
         image: {{ template "wordpress.image" . }}
         imagePullPolicy: {{ .Values.image.pullPol
         env:
         - name: ALLOW_EMPTY_PASSWORD
           value: {{ ternary "yes" "no" .Values.al
quote }}
         - name: MARIADB_HOST
         {{- if .Values.mariadb.enabled }}
           value: {{ template "mariadb.fullname" .
         {{- else }}
           value: {{ .Values.externalDatabase.host
         {{- end }}
         - name: MARIADB_PORT_NUMBER
         {{- if .Values.mariadb.enabled }}
           value: "3306"
         {{- else }}
           value: {{ .Values.externalDatabase.port
         {{- end }}
         - name: WORDPRESS_DATABASE_NAME
         {{- if .Values.mariadb.enabled }}
           value: {{ .Values.mariadb.db.name | quo
         {{- else }}
           value: {{ .Values.externalDatabase.data
         {{- end }}
         - name: WORDPRESS_DATABASE_USER
         {{- if .Values.mariadb.enabled }}
           value: {{ .Values.mariadb.db.user | quo
         {{- else }}
           value: {{ .Values.externalDatabase.user
         {{- end }}
         - name: WORDPRESS_DATABASE_PASSWORD
           valueFrom:
             secretKeyRef:
```

```
         - name: WORDPRESS_USERNAME
           value: {{ .Values.wordpressUsername | quote }}
         - name: WORDPRESS_PASSWORD
           valueFrom:
             secretKeyRef:
               name: {{ template "wordpress.fullname" . }}
               key: wordpress-password
         - name: WORDPRESS_EMAIL
           value: {{ .Values.wordpressEmail | quote }}
         - name: WORDPRESS_FIRST_NAME
           value: {{ .Values.wordpressFirstName | quote }}
         - name: WORDPRESS_LAST_NAME
           value: {{ .Values.wordpressLastName | quote }}
         - name: WORDPRESS_HTACCESS_OVERRIDE_NONE
           value: {{ ternary "yes" "no" .Values.allowOverri
quote }}
         - name: WORDPRESS_BLOG_NAME
           value: {{ .Values.wordpressBlogName | quote }}
         - name: WORDPRESS_SKIP_INSTALL
           value: {{ ternary "yes" "no" .Values.wordpressSk
| quote }}
         - name: WORDPRESS_TABLE_PREFIX
           value: {{ .Values.wordpressTablePrefix | quote }
         {{- if .Values.smtpHost }}
         - name: SMTP_HOST
           value: {{ .Values.smtpHost | quote }}
         {{- end }}
         {{- if .Values.smtpPort }}
         - name: SMTP_PORT
           value: {{ .Values.smtpPort | quote }}
         {{- end }}
         {{- if .Values.smtpUser }}
         - name: SMTP_USER
           value: {{ .Values.smtpUser | quote }}
         {{- end }}
         {{- if .Values.smtpPassword }}
         - name: SMTP_PASSWORD
           valueFrom:
             secretKeyRef:
```

```
        volumeMounts
         - mountPath
           name: wor
           subPath: w
         {{- if and
.Values.customHTAcc
         - mountPath
           name: cus
           subPath: w
         {{- end }}
         resources:
{{ toYaml .Values.re
{{- if .Values.metri
       - name: metric
         image: {{ te
         imagePullPo
quote }}
         command: [
'http://status.loca
         ports:
         - name: metr
           containerP
         livenessProb
           httpGet:
             path: /
             port: m
           initialDel
           timeoutSec
         readinessPro
           httpGet:
             path: /
             port: m
           initialDel
           timeoutSec
         resources:
{{ toYaml .Values.me
```

Still an Afterthought...

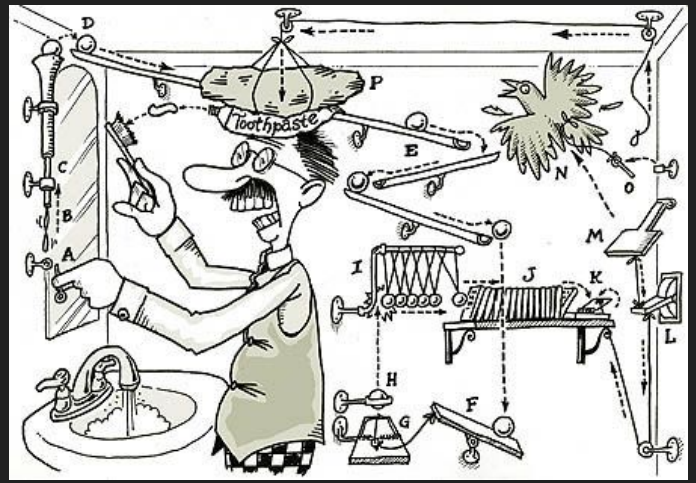
For applications: 

For infrastructure:

CLI	Language
aws/azure/gcp	Bash/YAML/JSON
terraform	HCL

For application infrastructure:

CLI	Language
kubectl	YAML
helm	YAML+Gotmpl
docker	YAML



✔️ Why Infrastructure as Code?

Standing on the Shoulders of Giants

💡🤔

What if we used real languages for infrastructure too?

Infrastructure as Code

Everything we **know and love** about languages:

- Tools (IDEs, test frameworks, linters).
- Abstraction and reuse.
- Ecosystems of libraries, sharing via package managers.
- Eliminate copy-and-paste and clumsy templating.

All of the **safety and robustness** of infrastructure as code:

- Any cloud.
- Reliably checkpoint states and recover from failure.
- Review and commit infrastructure changes like code.
- Automate deployments on Days 1, 2, and beyond.

pulumi

Demo

Infrastructure as Code

Multi-Language Runtime

Many Clouds

MANY-CLOUD FRAMEWORK
Create modern cloud native applications that can run anywhere.

PRODUCTIVITY LIBRARIES
Libraries for best practices and productivity.

FOUNDATION PROVIDERS
Unopinionated support for all clouds and their resources.

Kubernetes as Code

```typescript
import * as gcp from "@pulumi/gcp";
import * as k8s from "@pulumi/kubernetes";
import { Config } from "@pulumi/pulumi";

// Provision a new GKE cluster.
let cluster = new gcp.container.Cluster("gke-cluster", {
    initialNodeCount: config.get("nodes") || 3,
    masterAuth: {
        username: "master",
        password: config.requireSecret("clusterPassword"),
    },
    nodeConfig: { machineType: "n1-standard-1" },
});

// Create a canary deployment to test that this cluster works.
const name = "testapp";
const canaryLabels = { app: `canary-${name}` };
const canary = new k8s.apps.v1.Deployment("canary", {
    spec: {
        selector: { matchLabels: canaryLabels },
        replicas: 1,
        template: {
            metadata: { labels: canaryLabels },
            spec: { containers: [{ name, image: "nginx" }] },
        },
    },
}, { provider: cluster.provider });

// Export the config so that clients can easily access our cluster.
export let kubeconfig = cluster.kubeconfig;
```

```typescript
import * as k8s from "@pulumi/kubernetes";
import * as azure from "@pulumi/azure";

// Create an Azure CosmosDB with MongoDB support.
const cosmosdb = new azure.cosmosdb.Account("cosmosDb", {
    kind: "MongoDB",
    enableAutomaticFailover: true,
    geoLocations: [
        { location: config.location, failoverPriority: 0 },
        { location: config.failoverLocation, failoverPriority: 1 }
    ]
});

// Put the Azure CosmosDB auth info into a Kubernetes secret.
const mongoConnStrings = new k8s.core.v1.Secret("mongo-secrets", {
    metadata: { name: "mongo-secrets" },
    data: parseConnString(cosmosdb.connectionStrings),
});

// Create our app using Azure CosmosDB instead of self-hosted MongoDB.
const app = new k8s.helm.v2.Chart("app", {
    repo: "bitnami",
    chart: "node",
    version: "4.0.1",
    values: {
        serviceType: "LoadBalancer",
        mongodb: { install: false },
        externaldb: { ssl: true, secretName: mongoConnStrings.name },
    }
});
```

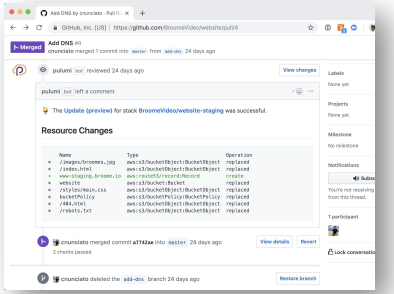
Demo

Kubernetes as Code

Automated Delivery

Most production deployments are triggered using automation.

- **GitOps**: Review changes in SCM, trigger deployments from CI/CD.
- **One Workflow**: Application containers, application config, infrastructure.



GitHub

ATLASSIAN

GitLab

Visual Studio Team Services

npm

Spinnaker

Jenkins

circleci

Travis CI

codefresh

kubernetes

aws

Google Cloud Platform

Azure

vmware vSphere

openstack.

DigitalOcean

```
~$ mocha ec2tests.js

  Infrastructure
    #server
      ✓ must have a name tag
      1) must not use userData (use an AMI instead)
    #group
      ✓ must not open port 22 (SSH) to the Internet


  2 passing (17ms)
  1 failing

  1) Infrastructure
       #server
         must not use userData (use an AMI instead):
     Error: Illegal use of userData on EC2 instance
        urn:pulumi:TestStack::TestPROJECT::aws:ec2/instance:Instance::web-server-www
      at pulumi.all.apply (ec2tests.js:21:26)
```

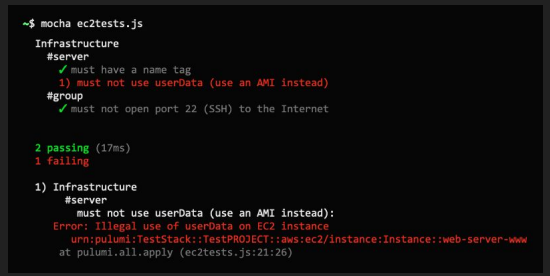
Cloud Engineering - Test Your Infrastructure

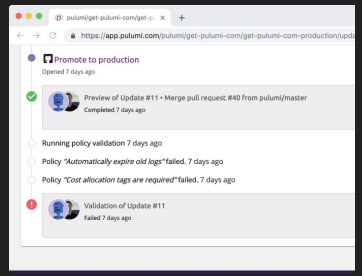
Unit testing: Ensure infrastructure configuration is correct.

Integration testing: Ensure the system works after deploying.

Enforce policy: Stop security issues before they ship.

More exotic testing: Fault injection, fuzzing, scale testing.





✔️ Why Infrastructure as Code?
✔️ Standing on the Shoulders of Giants

The Future of Cloud Architectures

Developers want to focus on business logic!

- **PaaS**: deploy small units of application-centric packaging (e.g, containers).
- **Serverless**: focus on application logic, pay per use, less fixed infrastructure.
- **NoCode**: for vertical applications, no code at all (powered by IaC underneath).

The world is powered by **infrastructure building blocks**.

Great things will come to those who can build bigger things out of smaller things.

Using real languages for infrastructure enables **a virtuous cycle of building**.

Teams where developers, infrastructure engineers, and operators collaborate will win.

In Summary

The power of real programming languages
with the safety and robustness of infrastructure as code.

It’s Just Code -- Have Some Fun!

Q&A

https://pulumi.io

@PulumiCorp

Joe Duffy
@funcOfJoe
Pulumi Founder and CEO