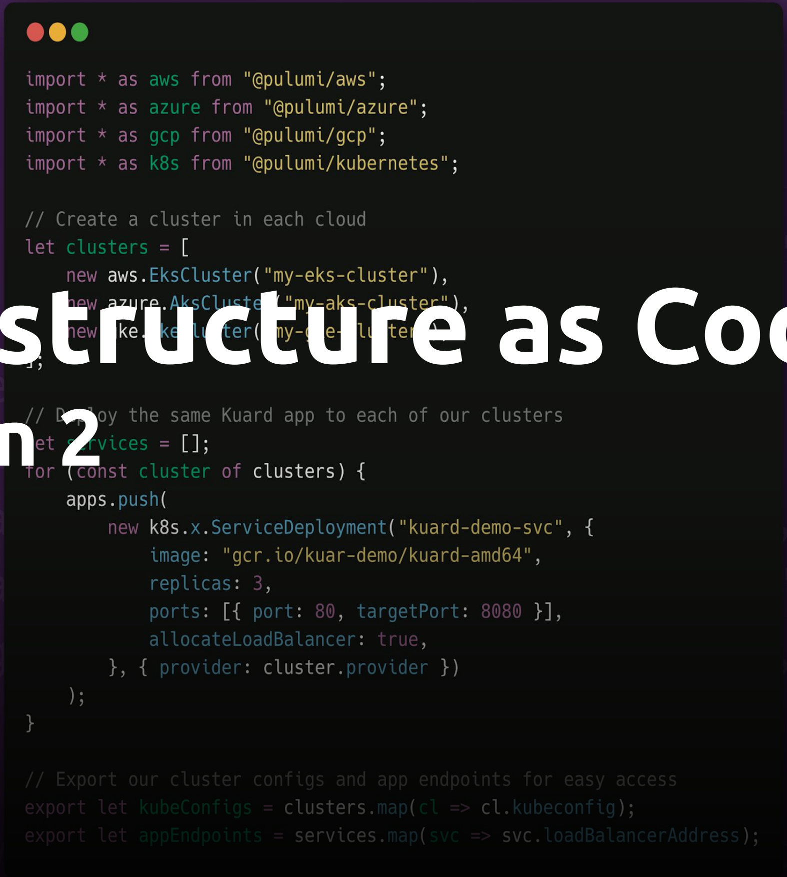
pulumi

Infrastructure as Code

Session 2



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

// Create a cluster in each cloud
let clusters = [
    new aws.EksCluster("my-eks-cluster"),
    new azure.AksCluster("my-aks-cluster"),
    new gke.GkeCluster("my-gke-cluster"),
];

// Deploy the same Kuard app to each of our clusters
let services = [];
for (const cluster of clusters) {
    apps.push(
        new k8s.x.ServiceDeployment("kuard-demo-svc", {
            image: "gcr.io/kuar-demo/kuard-amd64",
            replicas: 3,
            ports: [{ port: 80, targetPort: 8080 }],
            allocateLoadBalancer: true,
        }, { provider: cluster.provider })
    );
}

// Export our cluster configs and app endpoints for easy access
export let kubeConfigs = clusters.map(cl => cl.kubeconfig);
export let appEndpoints = services.map(svc => svc.loadBalancerAddress);
```

Session 2 Overview

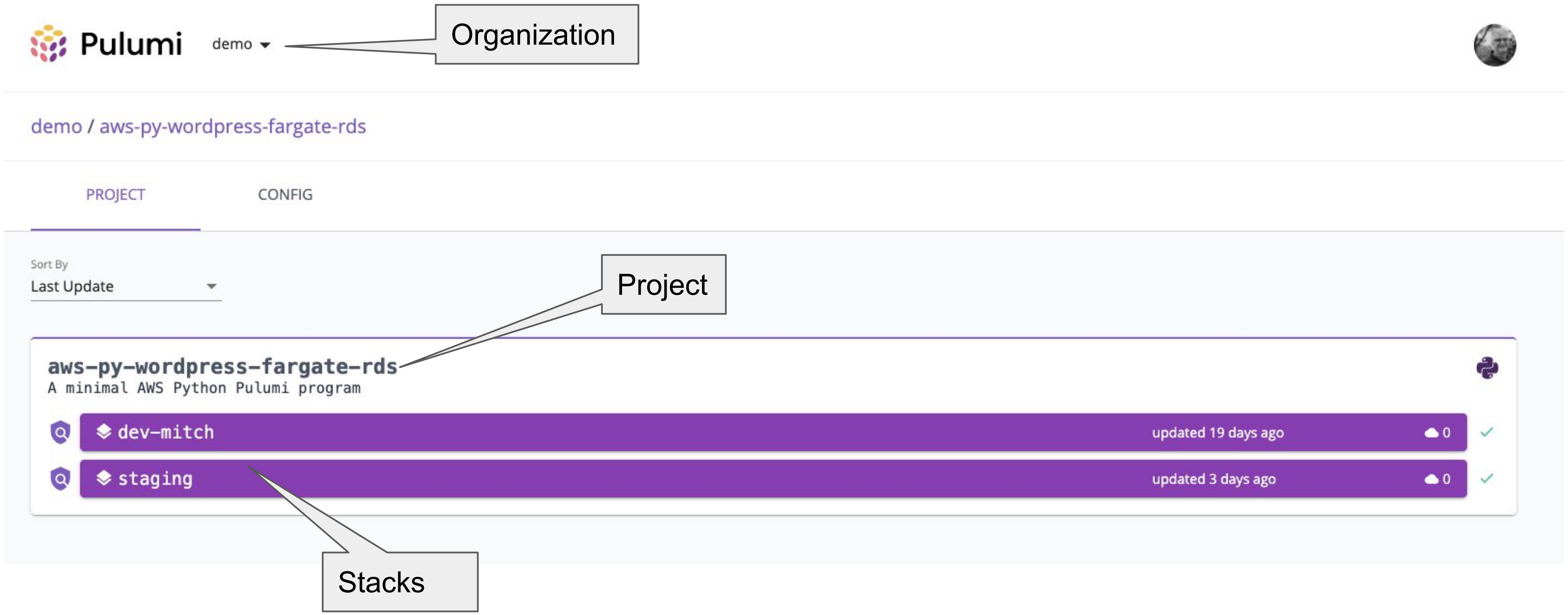
- Pulumi 101 (slides) - 30 mins
  - Projects, Stacks & Resources
  - Inputs and outputs and secrets
  - Stack Configuration
- Creating stacks and resources (hands-on) - 45 mins
- Wrap up and Q&A - 15 mins

Project, Stacks, and Resources

Projects and Stacks

- Projects = Pulumi infrastructure as code (i.e. resource declarations)
- Stacks = Instantiations of a project.
    - Can have any number of stacks for a project (e.g. dev, staging, preprod, prod, etc).
- Stacks can be driven by stack-specific configuration files.
    - E.g. dev stack is driven by Pulumi.dev.yaml which may deploy to region X,
    - While prod stack is driven by Pulumi.prod.yaml which may deploy to region Y and deploy with a specific number of instances or pods or whatever.
- Stack Names
    - Relative stack name: “dev” or “prod”
        - Related to the stack config file
    - Full stack name: “<org>/<project>/<stack>”
        - E.g. acme/ecommerce/staging

Projects and Stacks in the UI



aws-py-wordpress-fargate-rds
A minimal AWS Python Pulumi program

dev-mitch	updated 19 days ago	0	✓
staging	updated 3 days ago	0	✓

Pulumi CLI - Stack Basics

- To instantiate or update a stack: **`pulumi up`**
- To destroy a stack: **`pulumi destroy`**
- To see the list of stacks: **`pulumi stack ls`**
  - The stack with the * (asterisk) is the currently active stack.
- To see the current stack's resources and info: **`pulumi stack`**
- To select a different stack: **`pulumi stack select`**
- To initialize a new stack: **`pulumi stack init`**
- To see a stack's outputs: **`pulumi output`**
  - To see secret outputs: **`pulumi output --show-secrets`**
- To refresh a stack state by querying the providers: **`pulumi refresh`**
  - Generally only done when something goes wrong.
  - **`pulumi up`** takes **`-r`** and **`--expect-no-changes`** flags

Doc Link: https://www.pulumi.com/docs/reference/cli/

Pulumi Stack Configuration

- To set configuration values: **`pulumi config set <KEY> <VALUE>`**
- To set secret configuration values: **`pulumi config set <KEY> --secret`**
    - Will then be prompted for the value.
- To use configuration values in code:

```
import pulumi
config = pulumi.Config()
num_vms = config.get("num_vms") or 1
username = config.require(“username”)
password = config.get_secret(“password”)
api_key = config.require_secret(“api_key”)
```

- The “get” functions will return undefined if config not found.
- The “require” functions fail if config not found.

Pulumi Resource Basics

- Main Documentation for Pulumi Providers:
  - https://www.pulumi.com/docs/reference/pkg/
- Resource Declaration Example:

```
resource_group = resources.ResourceGroup(“resource_group”)
```

  - Creates a resource group with a name like “resource_group24d4d”
    - Autonaming is default behavior to avoid naming conflicts.
    - Generally this is fine since other resources can reference the name property.
  - Can override autonaming behavior by specifying “name” property:

```
resource_group = resources.ResourceGroup(“resource_group”, resource_group_name=”my_rg”)
```

    - Creates a resource group with name “my_rg”

Resource Options

- Resources Options are a set of properties that Pulumi makes available to modify default behaviors.
    - **`parent`**: establish a parent-child relationship. Mostly used in “component resources”
    - **`dependsOn`**: specify an explicit dependency.
    - **`ignoreChanges`**: mark that certain properties should be ignored when deciding whether or not to update.
    - **`protect`**: Mark a resource a protected. Pulumi will not allow the resource to be destroyed.
    - **`provider`**: pass an explicit provider.
    - **`additionalSecretOutputs`**: specify properties that must be treated as secrets.
    - **`aliases`**: used for refactoring where a resource name or parent path changes so that Pulumi doesn’t see it as a replacement action (i.e. create new/delete old).

Doc Link: https://www.pulumi.com/docs/intro/concepts/resources/#options

Accessing Resource Outputs

- Some resource properties are not known until the resource is deployed.
  - When referenced as inputs for other resources, Pulumi generally takes care of waiting.
  - https://www.pulumi.com/docs/intro/concepts/inputs-outputs/
- "apply" - Used for a single property.

```
url = virtual_machine.dns_name.apply(
    lambda dns_name: "https://" + dns_name
)
```

- "all" - Used when need to wait for multiple properties

```
connection_string = Output.all(server=sql_server.name, db=database.name) \
    .apply(lambda args: f"Server=tcp:{args['server']}.database.windows.net;initial
catalog={args['db']}...")
```

- "concat" - Easy way to create strings with promised values

```
url = Output.concat("http://", vm.dns_name, ":", iface.port, "/")
```

Secrets in Pulumi

Any value marked or identified as a secret is prevented from being written in logs or available in the SaaS or appear in plaintext in the outputs.

- Configuration file secrets (review)
  - Set secret in config: `pulumi config set <VALUE> --secret`
  - Get config secret in code: `password = config.get_secret(“password”)`
- Secrets in Code
  - Any value can be set as a secret in code:
    `api_key = pulumi.Output.secret(service.key)`
    - The api_key value can be used in the code but Pulumi treats it and anything that uses it as a secret.
- Secrets in Output - to see outputs marked as “[secret]” use --show-secrets option
  `Pulumi stack output --show-secrets`

Exercises

Prerequisites

Goal: Have a working Pulumi-Azure-Python environment

- Install Pulumi
  - Mac: **`brew install pulumi`**
  - Windows: **`choco install pulumi`**
  - Mac|Linux: **`curl -fsSL https://get.pulumi.com | sh`**
- Install Python3 (3.6+)
- Install az command line and login
  - **`az login`**
- (Optional test) Create a folder, pulumi new, pulumi up, pulumi destroy
  - **`mkdir test-project && cd test-project`**
  - **`pulumi new azure-python`**
  - **`pulumi up`**
  - **`pulumi destroy`**

Hands-On

Goal: Understand projects, stacks, and stack configuration

- Creating stacks and resources (hands-on) - 45 mins
  - Project/Stack Basics:
    - Stack outputs
    - Stack-specific configuration - plaintext
  - Advanced Project/Stack Concepts
    - Stack-specific configuration - secrets
    - Programmatic secrets in code and in outputs
    - Simple modules

Exercises - Stack Basics

Goal: Pulumi Stack Basics

- Set up workspace
  - `pulumi new https://github.com/pulumi/training/tree/main/azure-python/1_stack-basics`
- Open __main__.py
  - Follow exercises in comments.

Exercises - Advanced Stack Topics

Goal: Pulumi Stack Advanced Topics

- Set up workspace
  - `pulumi new https://github.com/pulumi/training/tree/main/azure-python/2_stack-advanced-topics`
- Open __main__.py
  - Follow exercises in comments.