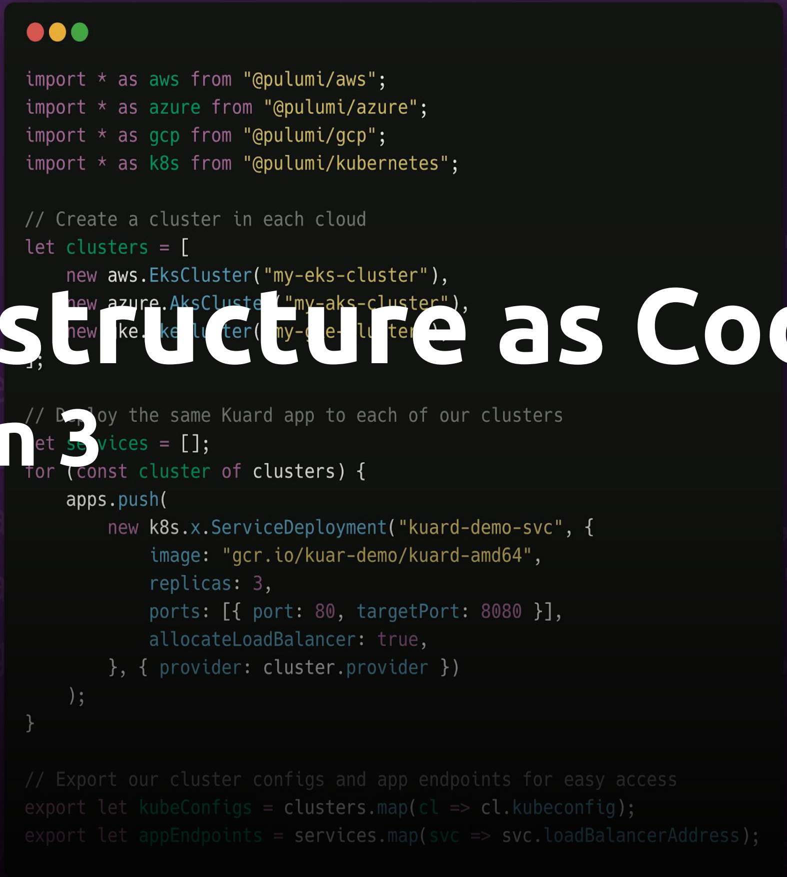
Infrastructure as Code

Session 3



```
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 3 Overview

- Questions from last time - 15 mins
- Architecting pulumi (slides) - 15 mins
    - Component resources
    - Stack references
- Hands-on architecting (hands-on) - 60 mins
    - Component resource
    - Stack references
- Best practices and troubleshooting (slides) - 15 mins
- Wrap up and Q&A - 15 mins

Component Resources

Component Resources

- Allows one to create a custom resource classes.
  - Abstract a set of resource declarations and logic into a single “resource”
  - Captures best practices.
  - Removes the need for everyone to understand all the nuances of given sets of resources

- Pulumi published packages are examples of this model.
  - E.g. Pulumi EKS package creates VPCs, subnets, IAM roles, EKS cluster all with one resource declaration.
    - EKS Package: https://www.pulumi.com/docs/reference/pkg/eks/

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

Resource Options (Review)

- Resources Options are a set of properties that Pulumi makes available to modify default behaviors.
  - ***`parent`****: establish a parent-child relationship.*
  - **`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.

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

Component Resources - Basic Structure

```
class MyResourceArgs:
    def __init__(self,
                 # name the arguments and their types (e.g. str, bool, etc)
                 parameter1:type,
                 parameter2:type):

        self.parameter1 = parameter1
        self.parameter2 = parameter2

class MyResource(pulumi.ComponentResource):
    def __init__(self,
                 name: str,
                 args: ClusterArgs,
                 opts: pulumi.ResourceOptions = None):

        super().__init__('custom:resource:MyResource, name, {}, opts)

        “””

           Declare resources, etc. (parent=self)
        “””

        # Used for display purposes.

        self.register_outputs({})
```

Stack References

Stack References

- Project/stack architecture may use different stacks for different parts of the solution.
  - “Stack 1”: Base VPC/Network stack
  - “Stack 2”: Services deployed on the base stack.
  - “Stack 2” needs to know, say, the network ID.

- Stack References allows stacks to retrieve outputs from other stacks.
  - Different stacks/projects can be written in different languages.
  - Does require some bookkeeping around the stack output names.
  - Bookkeeping can be mitigated using config values or the automation API.

Doc Link: https://www.pulumi.com/docs/intro/concepts/stack/#stackreferences

Stack References - Basic Usage

```
base_infra_stack = pulumi.StackReference(“acme/base_infra/dev”)

resource_group_name = base_infra_stack.get_output("resource_group_name")


storage_account = storage.StorageAccount(
    "appservicesa",
    resource_group_name=resource_group_name,
    kind=storage.Kind.STORAGE_V2,
    sku=storage.SkuArgs(name=storage.SkuName.STANDARD_LRS))
```

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 - Component Resource

Goal: Create and Use a Component Resource

- Component Resource (hands-on) - 30 mins
  - Create a Component Resource
  - Deploy the stack.
  - Modify code to add a “protect” flag.
- Set up workspace
  - **`mkdir comp-resource && cd comp-resource`**
  - **`pulumi new`** **https://github.com/pulumi/training/tree/main/azure-python/3_component-resources**
- Open __main__.py
  - Follow exercises in comments.

Hands-On - Stack References

Goal: Use Stack References

- Stack References (hands-on) - 30 mins
  - Modify “app” project to get the kubeconfig from the “base_cluster” stack.
- Create a directory with a couple of subdirectories - one for each stack.
  - **`mkdir session-3 && cd session-3`**
  - **`mkdir base_cluster app`**
  - **`cd base_cluster`**
    - **`pulumi new https://github.com/pulumi/training/tree/main/azure-python/4_stack-references/base_cluster`**
    - **`pulumi up`**
  - **`cd ../app`**
    - **`pulumi new https://github.com/pulumi/training/tree/main/azure-python/4_stack-references/app`**
- Modify app project code to use a stack reference to get the kubeconfig from the base_cluster.
- Deploy the app stack.

Best Practices and Troubleshooting

Programming Best Practices Apply to Pulumi

Pulumi uses programming languages, so programming best practices can be used with Pulumi code like any other code.

- Modularization
  - Basic code modules
  - Component Resources
- GitOps and Testing
  - Run Infrastructure as code through the same processes and tools as “traditiona” application code.
- Pin package versions
  - Don’t necessarily take the latest version of a pulumi sdk.

Organizing Projects and Stacks

- No “right” answer - there is a gamut of tradeoffs
- Monolithic vs multistack architectures
  - Monolithic is a natural place to start
    - **`alias`** resource option can help with refactoring.
      - Nice blog on the matter: https://www.pulumi.com/blog/cumundi-guest-post/
  - Microstacks or multi-stack architectures make sense if
    - Different/disparate teams at work.
    - Rate and nature of change is substantially different.
      - K8s stack and then service-specific stacks.
- Git repo or CI/CD pipeline alignment.
  - Existing pipelines may drive towards aligning Pulumi projects with those pipelines.

Policy as Code

- Pulumi Crosswalk (i.e. Policy as Code)
- Allows policies to be defined to protect against bad practices.
  - Based on the provider SDKs so all properties can be checked.
- Can be applied to all stacks or specific stacks.
- Can be advisory or mandatory
  - Advisory allows the stack deployment to continue.
  - Mandatory prevents the deployment.
- Are automatically applied across an organization.
  - Pulumi org admins can publish policy packs and make policy groups.
- Policy code can be written in a different language than stack.

Troubleshooting

- Sometimes things happen.
    - E.g. you terminate a `pulumi up` before it completes. This can result in a bit of a wonky state.
- State File Issues:
    - **`pulumi stack export | pulumi stack import`**
        - Will often clean up the state - especially in the case of an interrupted pulumi action.
    - **`pulumi refresh`**
        - Update the state with the current information from the resource providers.
        - Comes in handy if someone makes a change to pulumi-managed resources outside of pulumi (e.g. via cloud provider portal)
- Debugging:
    - Verbose debugging
        - **`pulumi up --logtostderr -v=9 2> out.txt`**
        - “-v” takes levels 1-9 with 9 being most verbose

Doc Link: https://www.pulumi.com/docs/troubleshooting/