Protecting and Segregating Resources in Pulumi

Introduction

When deploying infrastructure, there may be some constraints:

- Different teams own different tiers or parts of the infrastructure.
- Different tiers of the infrastructure are updated on different cadences.
- Stateful parts of the infrastructure (e.g. databases) need to be protected against accidental deletion.

Pulumi addresses the requirements via stack references, RBAC and resource protection flags.

Pulumi Projects, Programs and Stacks

Before discussing the solutions, it's important to understand some Pulumi constructs: projects, programs and stacks.

The Pulumi project and program is the code base being used to define a set of resources. For example, I may have a project (folder) that contains a program that declares an Azure Resource Group and CosmosDB and maybe some additional related infrastructure. This code base is managed in a git repository or similar software management solution.

The Pulumi stack is an instantiation of those resources based on a project/program. One can have any number of stacks that use the same project/program. Each stack will (likely) have a corresponding set of stack configuration that the program consumes when creating the resources. When the program runs it creates the resources as part of a given stack which is managed by Pulumi via the stack's state file. Stacks generally have some outputs which can then be consumed by humans or other systems. In the example described above, these outputs could be the resource group name and the CosmosDB document endpoint, etc.

In Pulumi Cloud, features like RBAC and deployments and the K8s operator are tied to Pulumi stacks. So the stack is a fundamental construct that drives the solutions described below.

Example Code

The discussion below references example code found here:
https://github.com/pulumi-demos/resource-protection-segregation

This repo has three Pulumi Projects:

- **BaseInfra**: Deploys the base stack on which the other two projects will deploy stacks.
- **App_InLine_StackRef**: Deploys a weblogic app on top of the BaseInfra using “in-line” stack reference.
- **App_ESC_StackRef**: Deploys a weblogic app on top of the BaseInfra using a Pulumi ESC environment to abstract and project the stack reference.

Stack References

Pulumi’s stack reference construct enables one stack to consume outputs from another stack. Given the CosmosDB project/stack mentioned above, maybe there are separate projects/stacks that deploy different apps and related infrastructure that need to push data into or retrieve data from the CosmosDB. In this case, the App stacks need to know the CosmosDB endpoint, etc. The App stacks can use stack references to programmatically obtain that information from the CosmosDB stack.

There are two ways to leverage stack references:

- Instantiate stack reference objects in the Pulumi program.
    - https://www.pulumi.com/docs/iac/concepts/stacks/#stackreferences
- Use the “stacks” provider in Pulumi ESC.
    - https://www.pulumi.com/docs/esc/integrations/infrastructure/pulumi-iac/pulumi-stacks/

Stack Reference in Pulumi Program

See:
https://github.com/pulumi-demos/resource-protection-segregation/tree/main/App_InLine_StackRef

Looking at the `config.py` file in that folder, one sees an example of an in-line stack reference to the BaseInfra stack. In a nutshell, a stack reference object is instantiated and then used to get the stack outputs from the BaseInfra stack.

Positives of this Approach:

- The code makes it clear and obvious that those values are coming from another stack.

Negatives of this Approach:

- The App project/stack has to know about the base project/stack. So there is some bookkeeping needed to pass this information to the App stack.
    - This means that if there are many “App” stacks using the same BaseInfra stack, then you need to copy that information to each App stack’s stack config file.
    - So, maintenance may become a challenge if the BaseInfra project name is changed.

- If you want to test the App stack, you either must have a BaseInfra stack deployed that can be referenced, or you need to modify the code to use config to temporarily pass in the values or even hardcode the values.

Stack Reference via Pulumi ESC Environment

See:
https://github.com/pulumi-demos/resource-protection-segregation/tree/main/App_ESC_StackRef

Looking at the `Pulumi.dev.yaml` file and `config.py` file in that folder, one sees an example of using a Pulumi ESC environment to project the BaseInfra stack outputs as stack config. In a nutshell, a Pulumi environment is created using the `pulumi-stacks` provider to consume the outputs from the BaseInfra stack and then project them as Pulumi stack config. So the the App project code itself just sees the BaseInfra stack outputs as config and doesn't even know the values are coming from another stack

Positives of this Approach:
- The App project/stack doesn't need to know about the base project/stack. So there is no bookkeeping needed to pass this information to the App stack.
    - You will have to specify the environment in each stack's stack config, but chances are the environment name will not change and with ESC composability it never has to.
- It's easy to test the App stack, since the stack sees the values as config. So, instead of having to have a BaseInfra stack or having to modify the coe, you can just add config to `Pulumi.STACK.yaml` for the different values which will override the values from the environment.

Negatives of this Approach:
- The fact that the values are coming from another stack are obfuscated. But this could also be seen as a positive.

Protecting Resources from Deletion

See:
https://github.com/pulumi-demos/resource-protection-segregation/blob/main/BaseInfra/__main__.py

The key concept for protecting resources in Pulumi is the `protect` Resource Option (https://www.pulumi.com/docs/iac/concepts/options/protect/). This resource option prevents Pulumi from performing an action that would result in the resource being deleted (or replaced - which implies a deletion).

In the code, linked above, the Resource Group resource has the `protect` option explicitly added. And then a Transform (https://www.pulumi.com/docs/iac/concepts/options/transforms/#stack-transforms) is used to set the `protect` flag for all the rest of resources and thus not require having to add the resource option to each resource explicitly.

If after running `pulumi up` for the stack, one tries to run `pulumi destroy` an error message will be seen along with information on how to remove the protection for a given resource. Generally, though if you indeed want to destroy the stack, you'll run "`pulumi state unprotect --all`" so as to remove protection for all the resources from the state and thus allow the `pulumi destroy` to be successful.