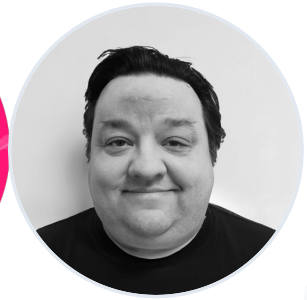
Working with AWS CloudFormation Stacks



Mike Brown
Senior Cloud Instructor

@mgbleeds

Globomantics Problem:

Manual changes to resources deployed by CloudFormation stacks are becoming inconsistent and mistakes are being made.

Making Updates

Manual changes should be avoided where possible

Changes to resources managed by CloudFormation should be made through CloudFormation

Stack Updates

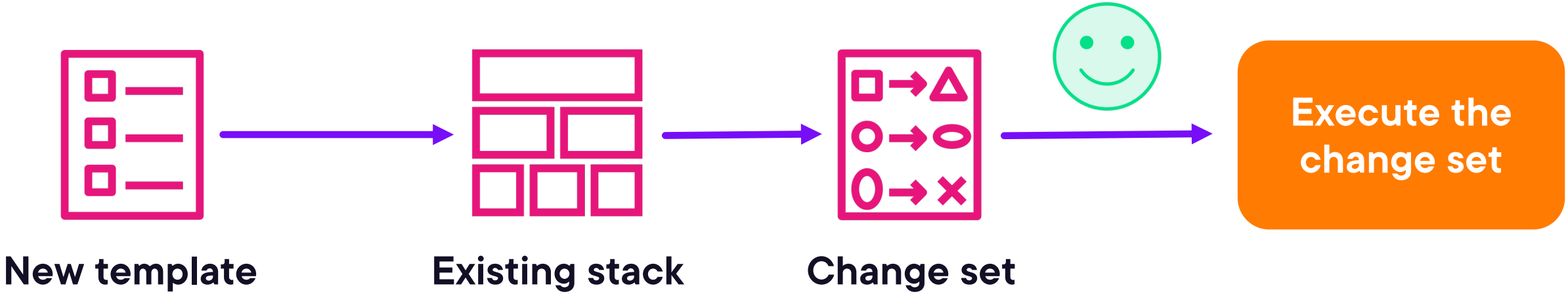
Update a new stack using a template with your new configuration

It would be good to preview changes before they are made

This allows you to assess the impact of changes to your resources

You can do this with CloudFormation change sets

Change Sets


Execute the change set
New template
Existing stack
Change set

Change Sets

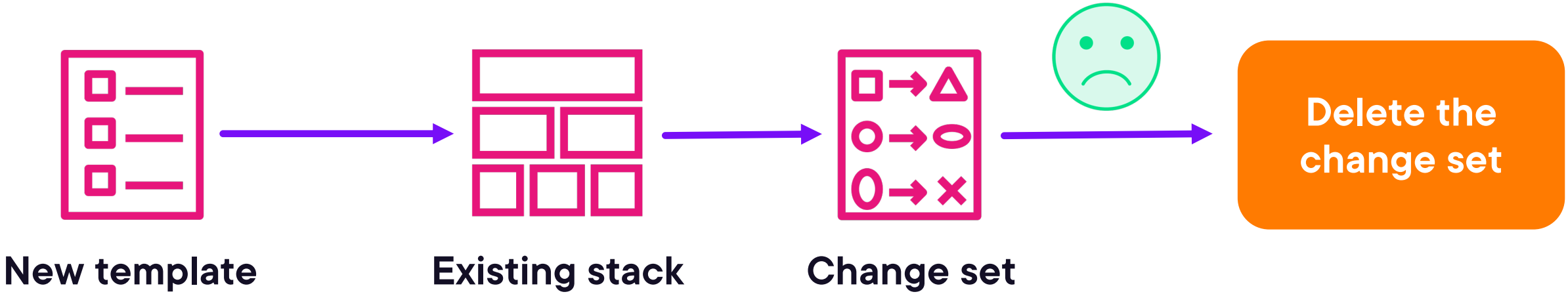
Reduce configuration mistakes

Reduce inconsistencies of configurations

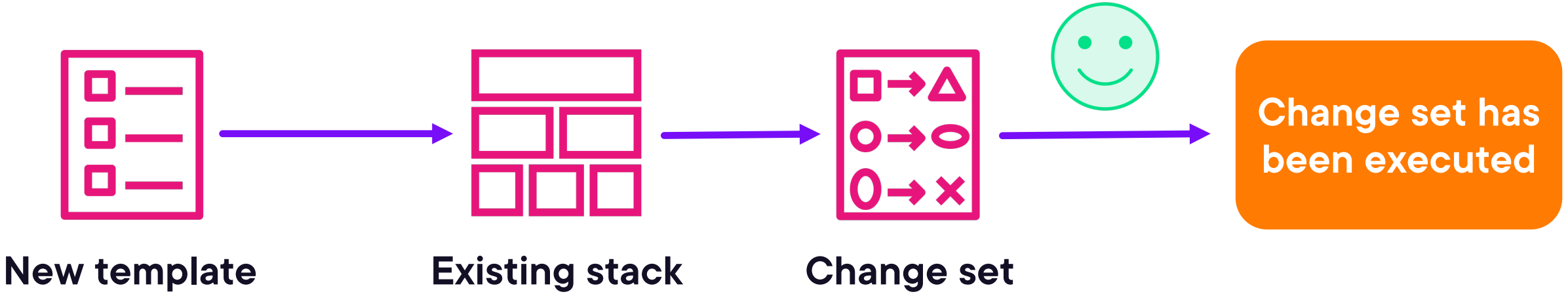
Speed up the update process

Enhance collaboration between different teams

Change Sets


Delete the change set
New template
Existing stack
Change set

Change Sets



Use AWS Config to see a timeline of all changes to your resources

DevOps Teams

Keep your templates in a version control system to track changes and help the review process

Use your CI/CD processes to automate the change set process

Change Sets

Do not indicate that changes will be carried out successfully

Do not assess permissions needed to make changes

Cannot assess the impact to resources that are not under the control of the stack being assessed

The AWS best practice is to use change sets for all stack updates.

AWS CloudFormation Best Practices

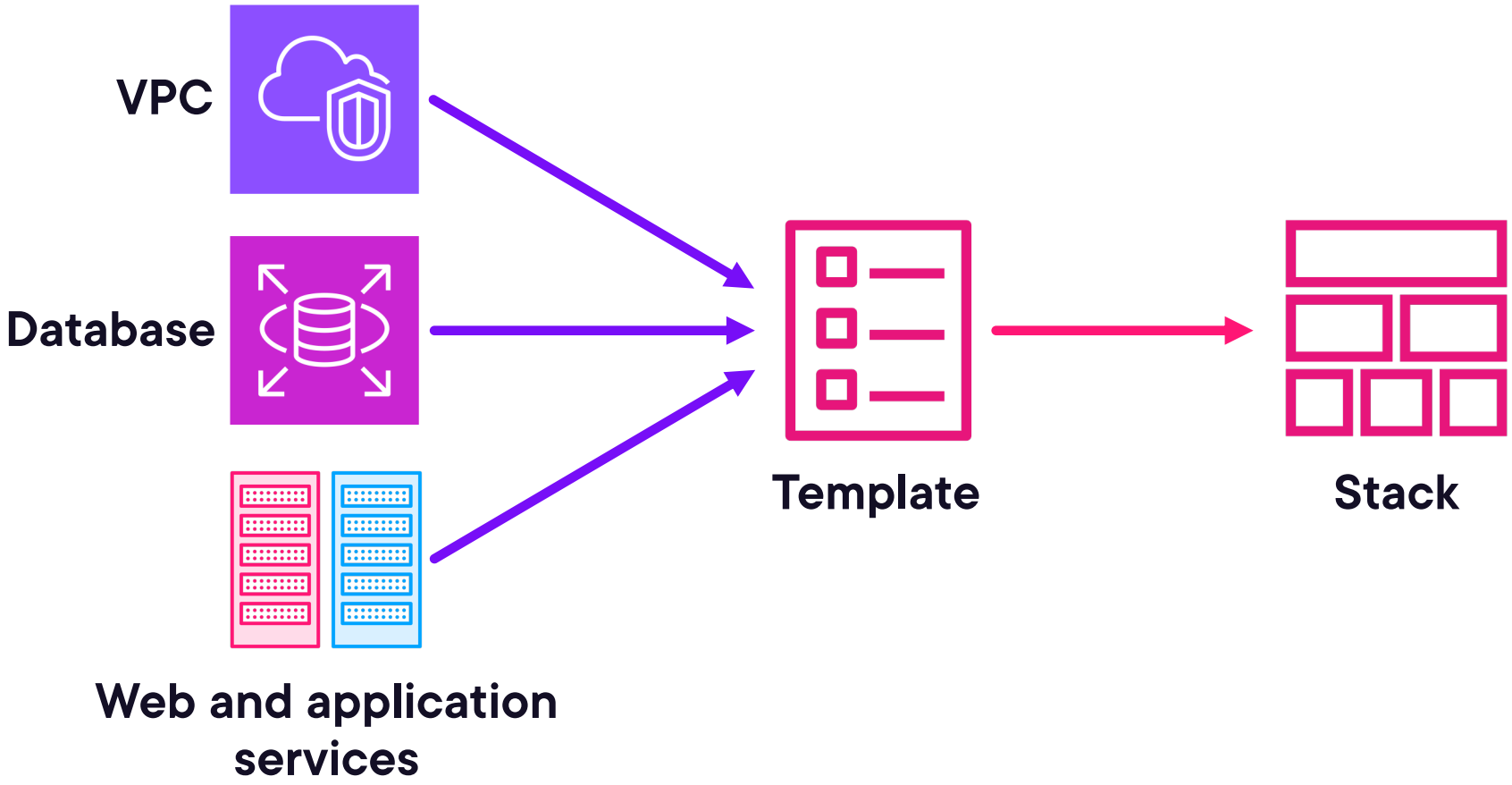
Make your templates reusable between regions, AWS accounts, and different workloads.

Making Templates Reusable

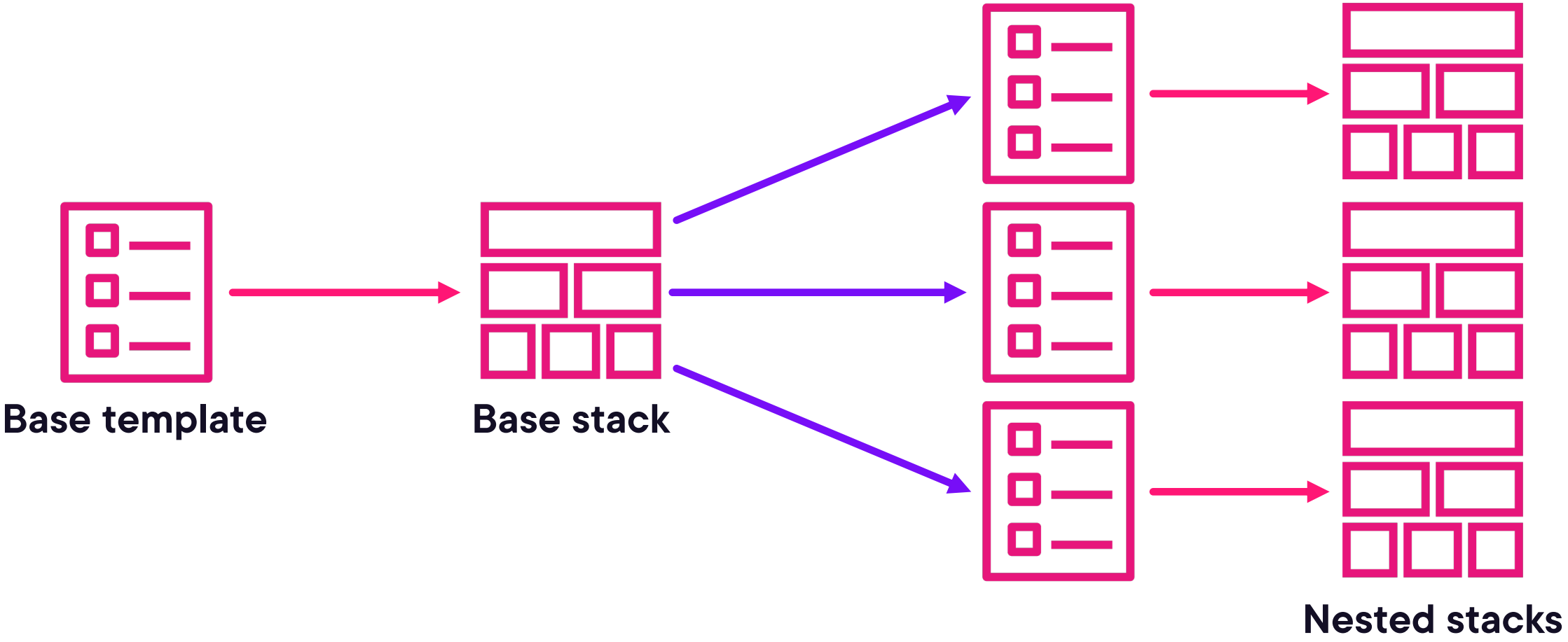
You should use parameters, mappings, and conditions whenever you can

Remove any hard-coded settings that might change in different stacks

Nested Stacks

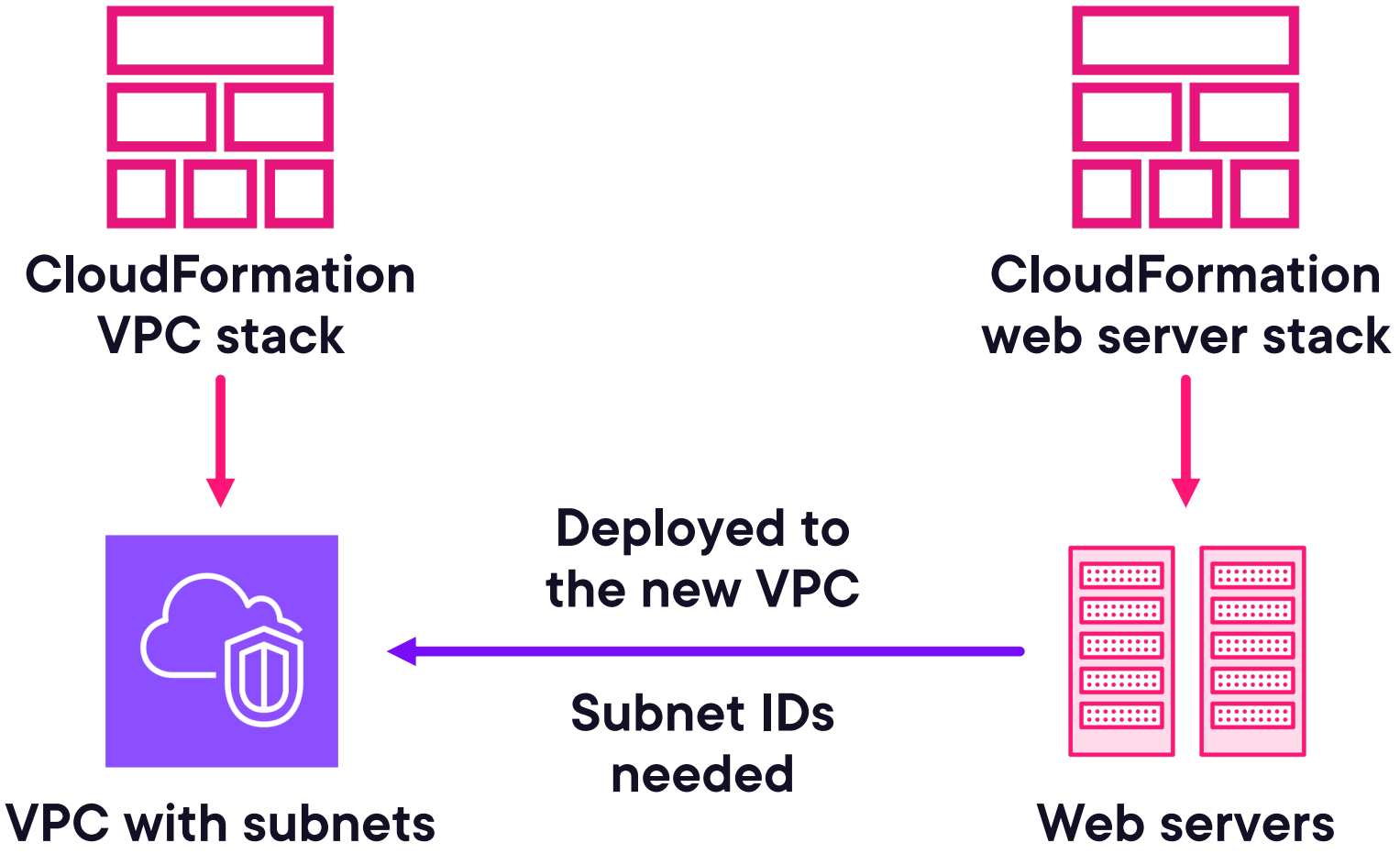


Nested Stacks



You break down complex infrastructure into smaller reusable templates that are often more generic and can be used repeatedly.

Cross-stack References



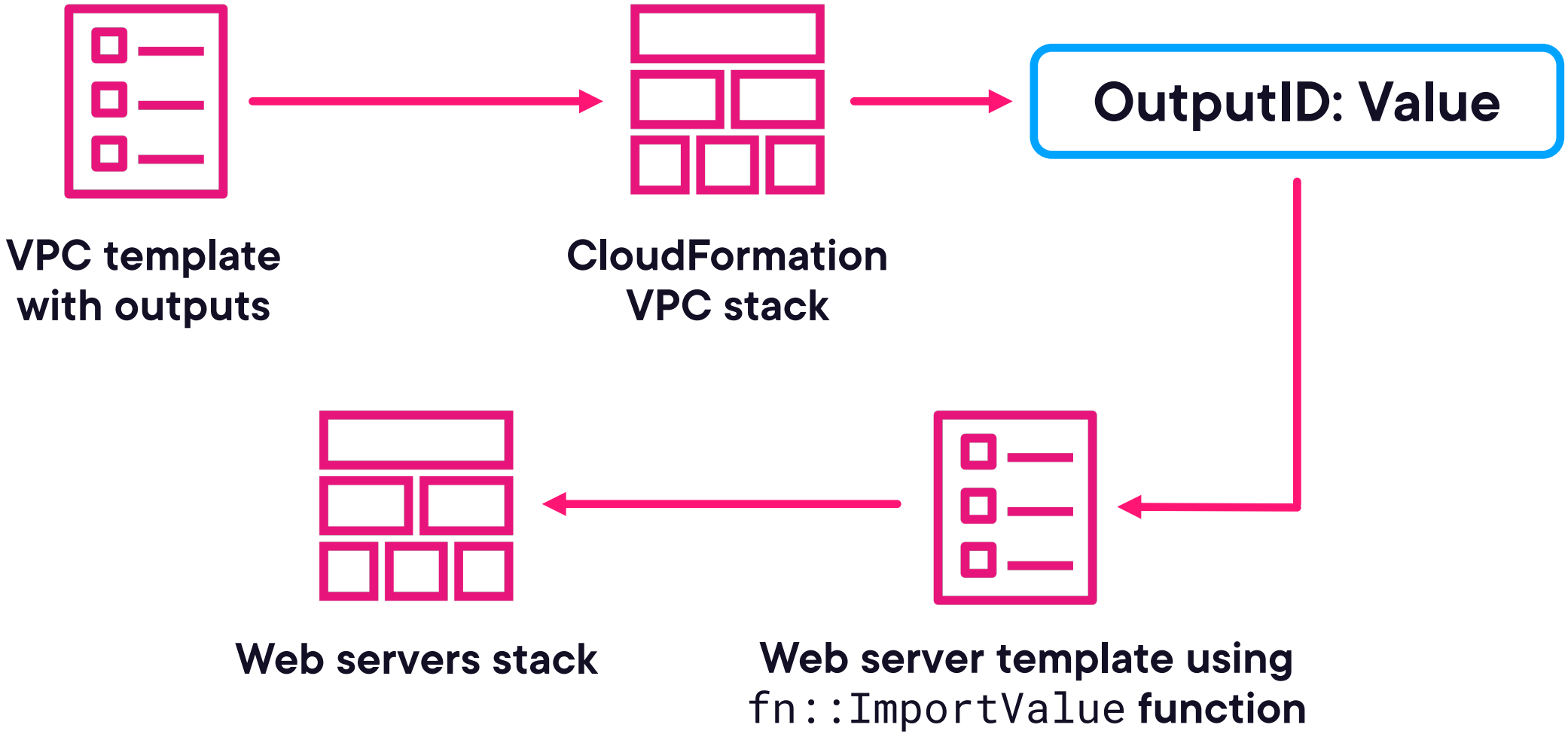
Cross-stack References

Wait until the VPC stack is created and locate the IDs of the new subnets

Update the web server template with the new IDs, and then create the web server stack

Use outputs and inputs instead

Cross-stack References



```
"Outputs" : {
    "StackSubnet" : {
        "Description" : "The ID of the Subnet",
        "Value" : { "Ref" : "mySubnet" },
        "Export" : {
            "Name" : {"Fn::Sub": "${AWS::StackName}-SubnetID" }
} } }
```

Example Outputs Section

This output section creates an output containing the subnet ID of a new subnet created by this stack. It assigns an ID to the output of `Stackname-SubnetID`**.**

```
"myEC2Instance" : {

      "Type" : "AWS::EC2::Instance",

      "Properties" : {

        "ImageId" : { "Fn::FindInMap" : [ "AMIRegionMap", { "Ref" : "AWS::Region" },
                                "windowswebserver" ]},

        "InstanceType" : "m5.large",

         "SubnetId" : {"Fn::ImportValue" : {"Fn::Sub" : "stack1-SubnetID"}

      }}}}
```

Example of the `Fn::ImportValue` Function

Instead of a hardcoded value for the `SubnetID` **property, we use** `Fn::ImportValue` **to import the value from the output named** `stack1-SubnetID`**. "Stack 1" is the name of the stack that created the output.**

Stack Policies

Give an extra level of protection for key resources

Prevent resources from being accidentally updated or deleted

When a stack is created, all update actions are allowed on all resources

Once a stack policy is set, all resources in the stack are protected

Specify “allow” statements for resources you want to allow updates to

CloudFormation Best Practices

Create CloudFormation modules

Organize stacks by lifecycle and ownership

Automate with CI/CD pipelines and implement other DevOps best practices

Enable logging and monitoring of stacks using AWS CloudTrail, AWS Config, and Amazon CloudWatch