Using GitOps to Automate Kubernetes Deployments with Flux 2

Understanding the Challenges of Automating Deployments

Nigel Brown

@n_brownuk www.windsock.io

The Problem with Automated Deployments

DevOps team adopts cloud native approach

Management's confidence is beginning to wane

A GitOps approach promises to improve delivery

Exemplified by:

- Containerized applications
- Kubernetes clusters

Reliability issues:

- Failed deployments
- Environment ambiguity

Improving reliability:

- Adopt GitOps techniques
- Flux automation tools

Module Outline

Coming up:

- How Kubernetes gets things done
- Cloud native application delivery
- Configuration drift

Container Orchestration

Orchestration is required to manage containers at scale

Kubernetes is the de facto delivery platform for containerized applications

Automation to the Rescue

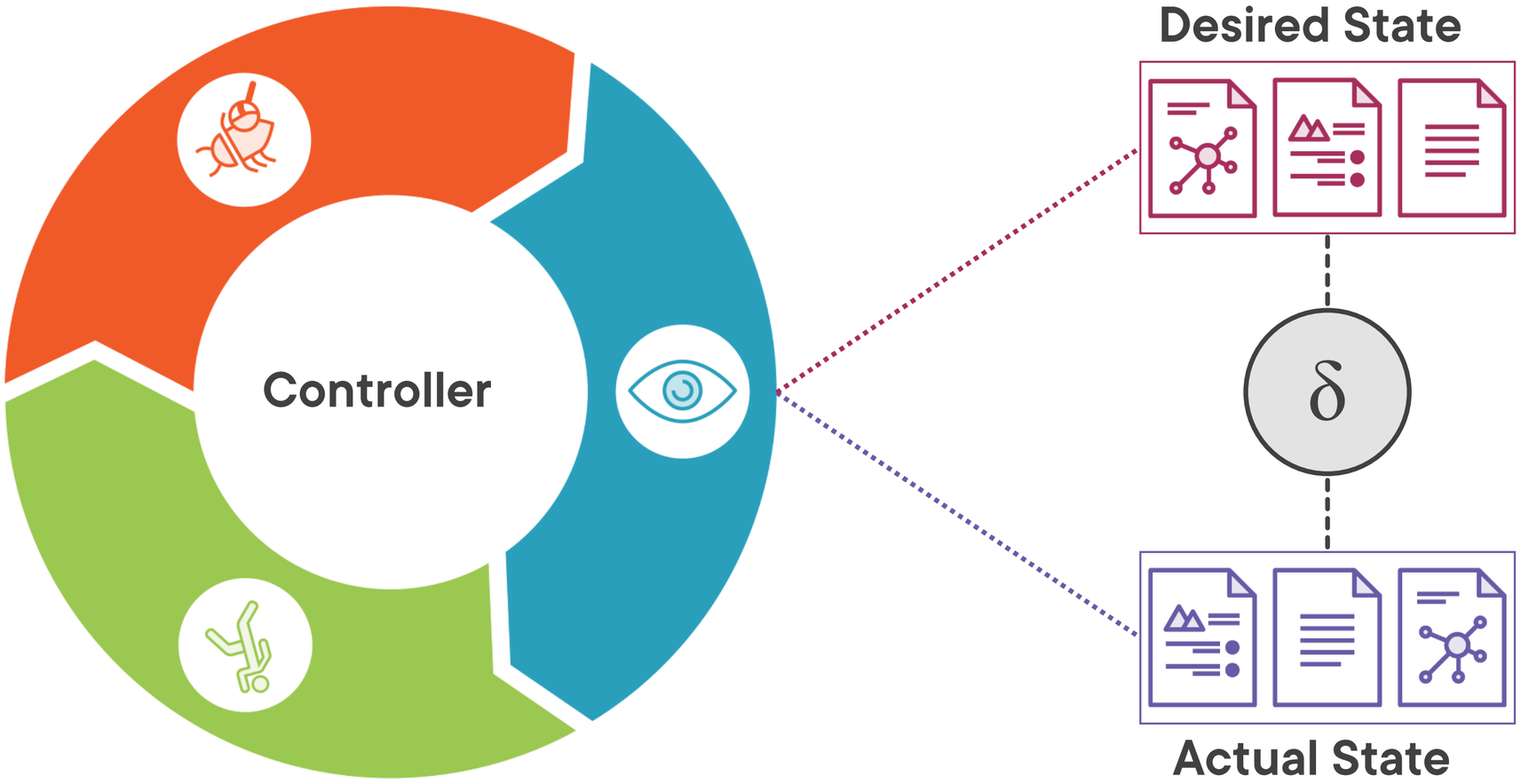
Rolling out and rolling back containerized workloads in the cluster

Enabling co-dependent workloads to discover each other

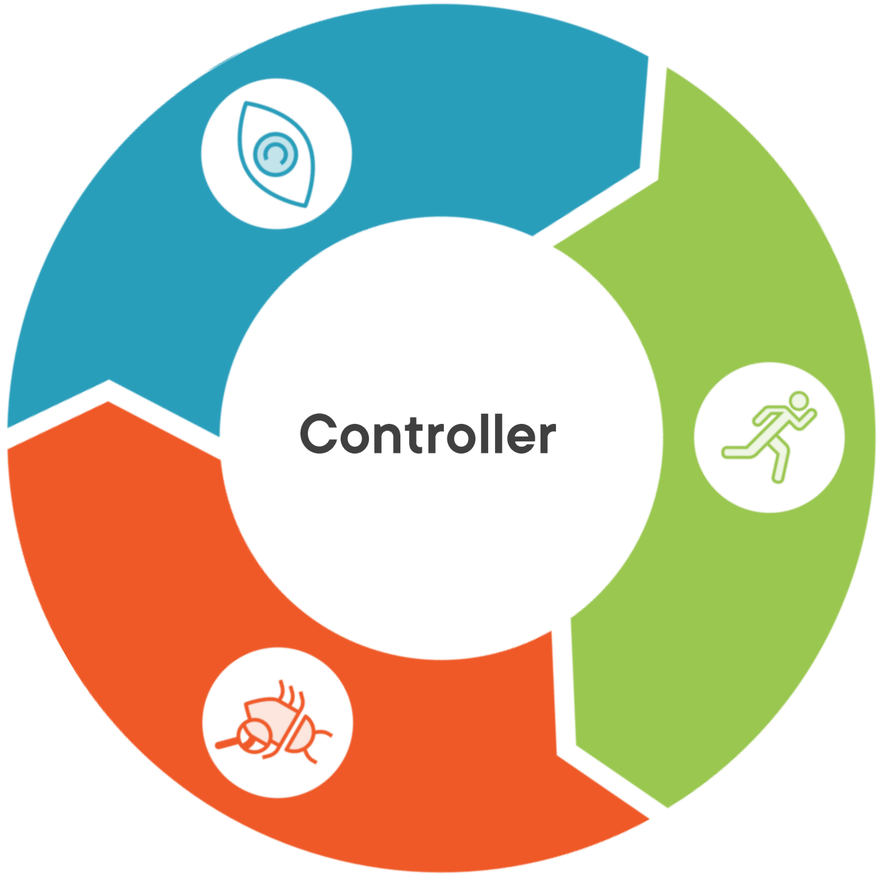
Re-starting deployed workloads when they enter a failed state

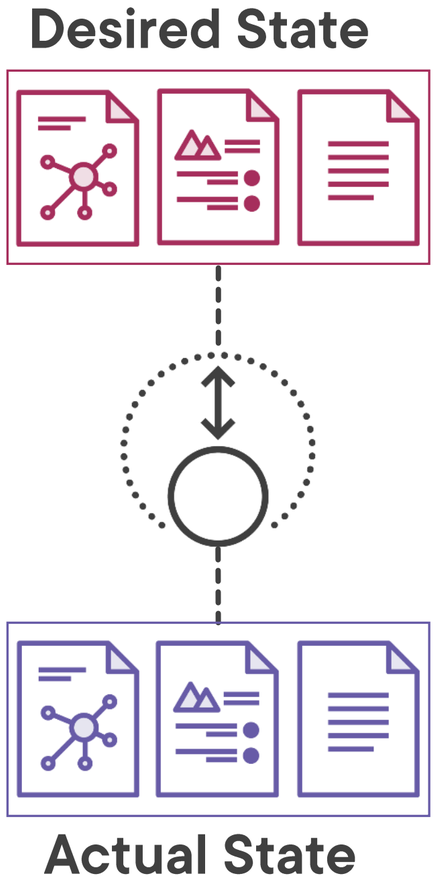
Automatic scaling of workloads in response to fluctuating demand

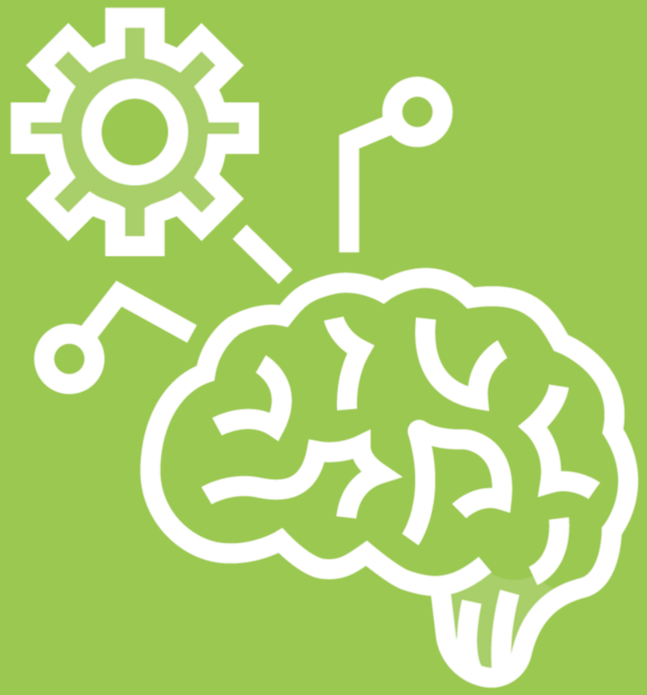
Desired State Reconciliation



Desired State Reconciliation



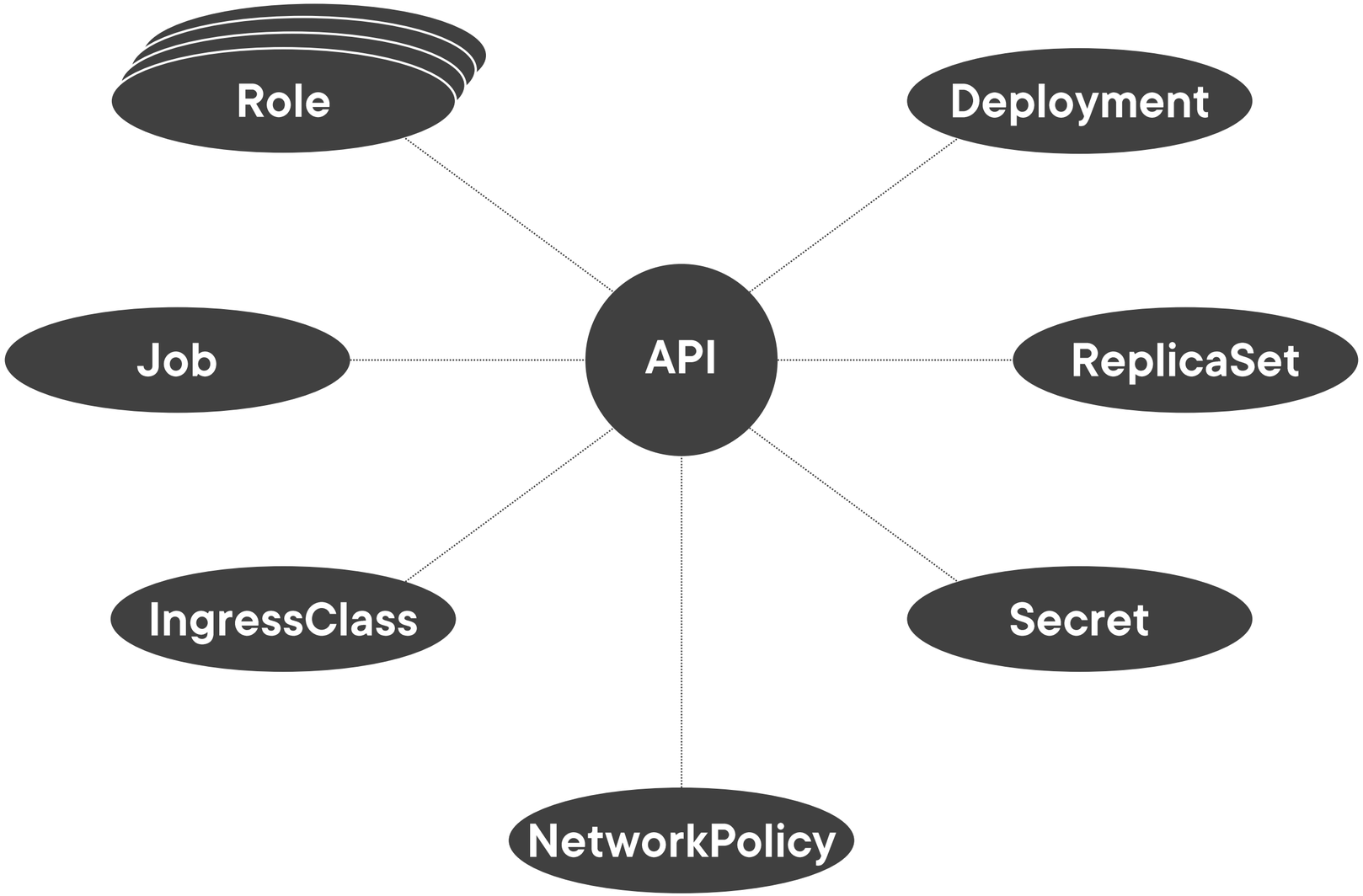




Desired State Reconciliation

How does Kubernetes know what the desired state of the system should be?

Kubernetes API

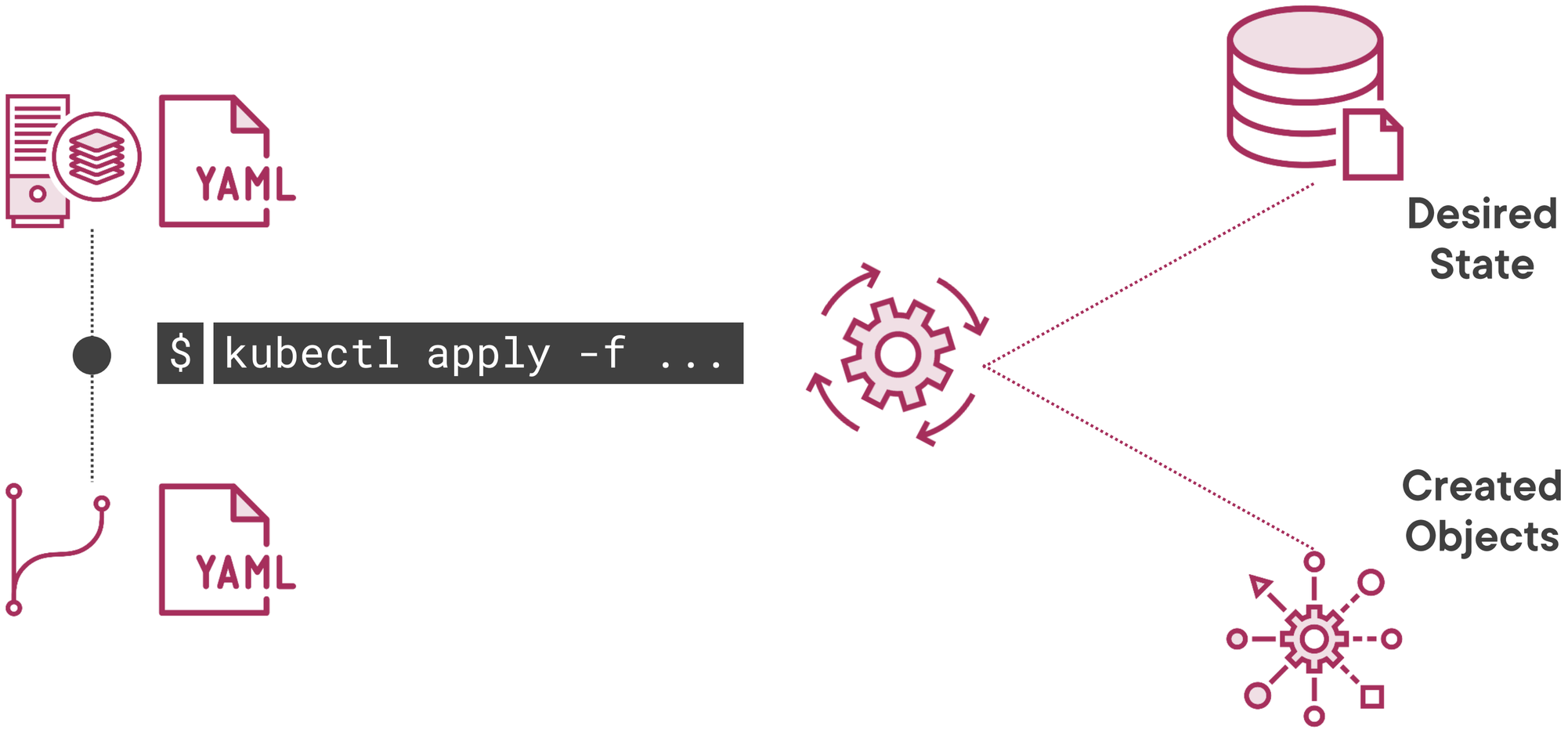


Declarative Configuration

pruner-job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pruner
spec:
  template:
    metadata:
      name: pruner
    spec:
      containers:
      - name: pruner
        image: mycorp/pruner:v1.2
        command:
        - “prune"
        - "-c"
        - “/etc/prune.conf”
      restartPolicy: Never
```

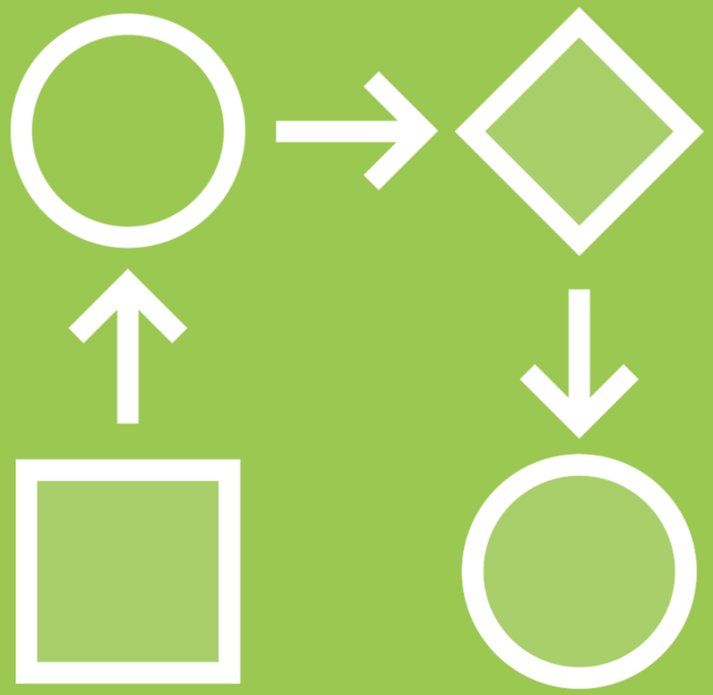
Applying Configuration



Cloud Native Workflows

Declarative configuration and desired state reconciliation are key components

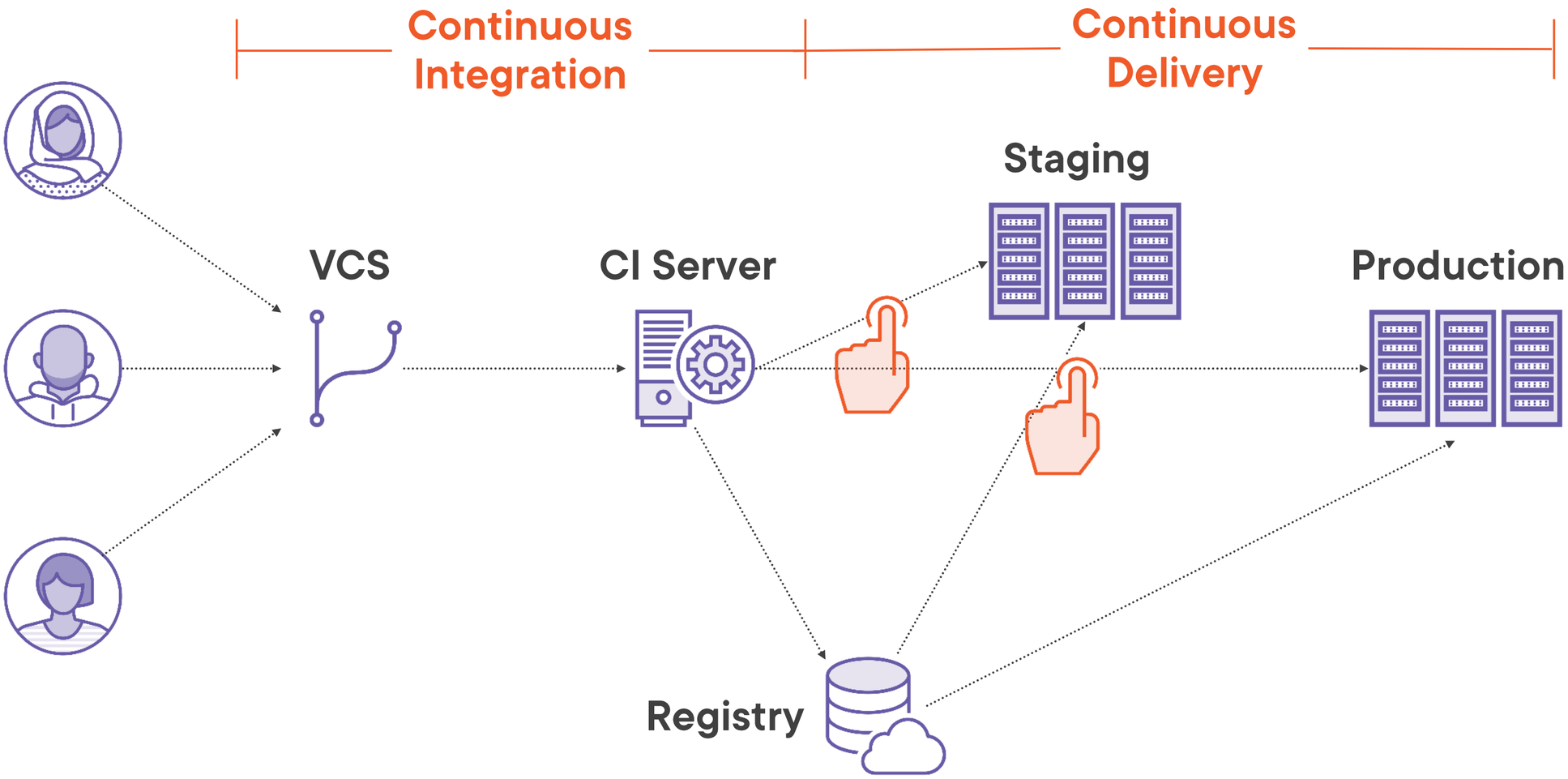
Cloud native workflows also play a big part in the life of our software applications



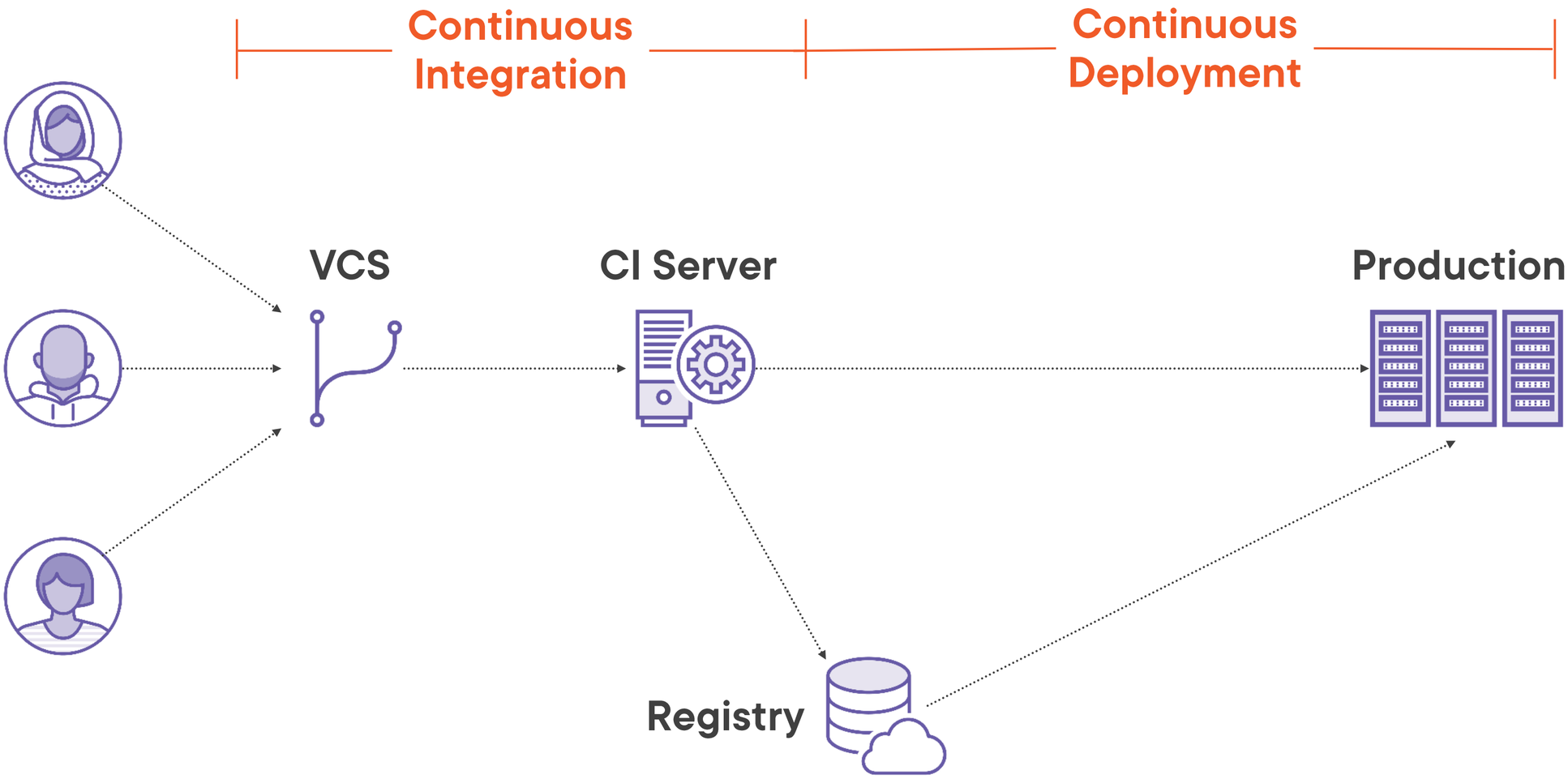
Cloud Native Workflows

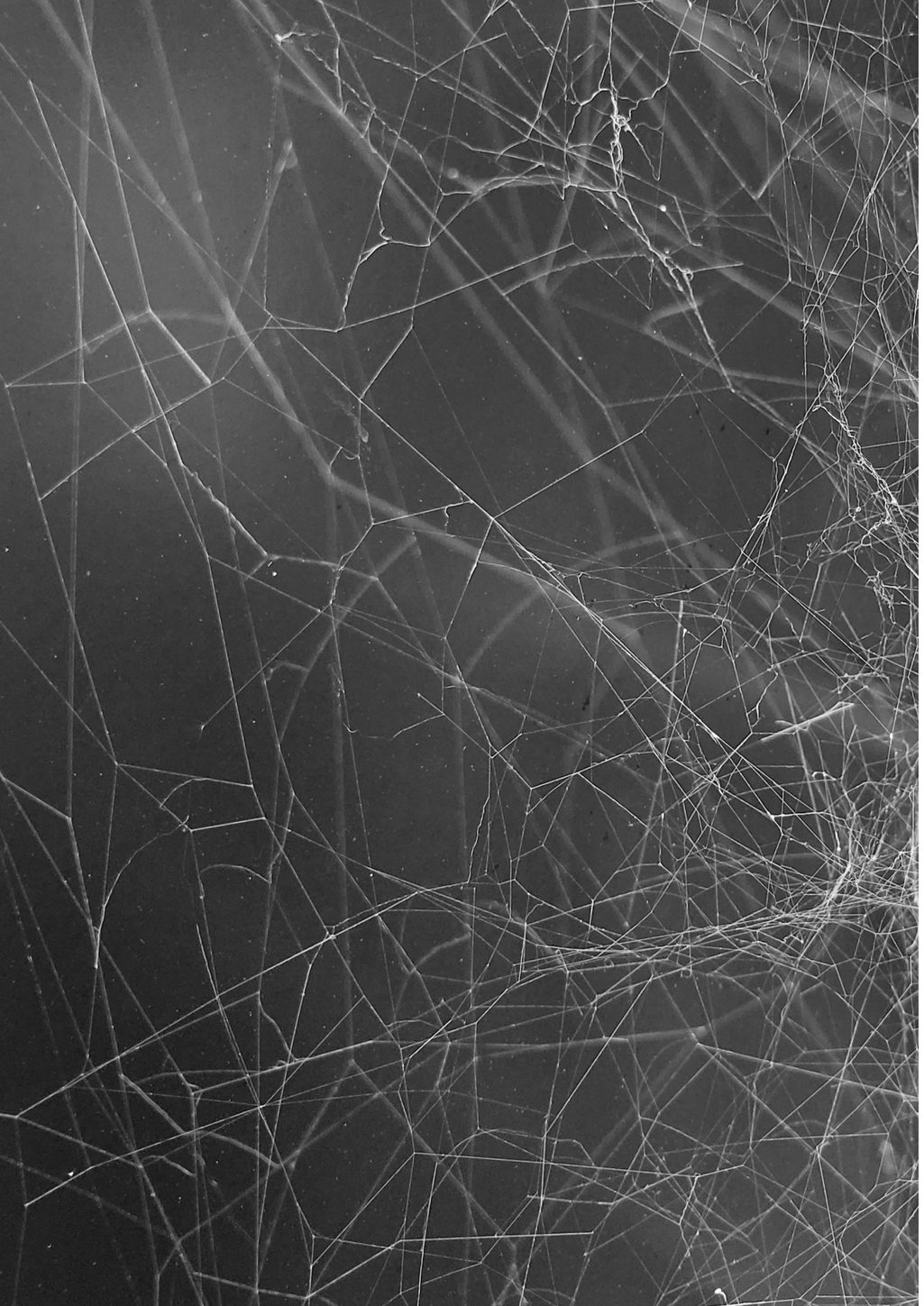
Automated workflows or pipelines are used to increase the speed, frequency, and quality of application service deployments.

Cloud Native Workflows



Cloud Native Workflows



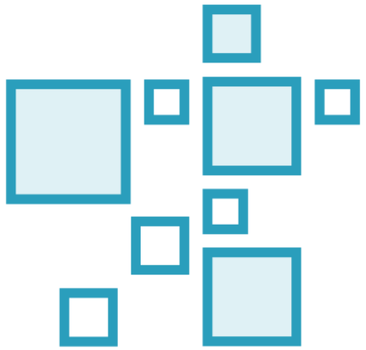


Simplified and distilled representation

- How does it work in a Kubernetes environment?
- What challenges might we encounter along the way?

Container Images

Container images encapsulate the deployable cloud native application service.



Code + Dependencies

Contains application code and the package or library dependencies.

Immutable

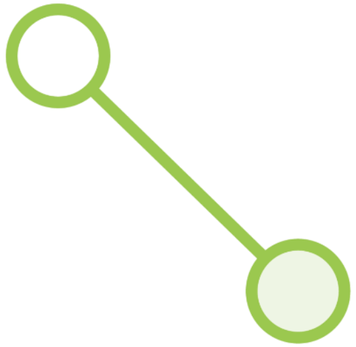
New versions of applications require a new container image.

Container Registry

Often stored in remote registries and require pulling before running.

Workload Configuration

Kubernetes config manifests are used to define the workload.

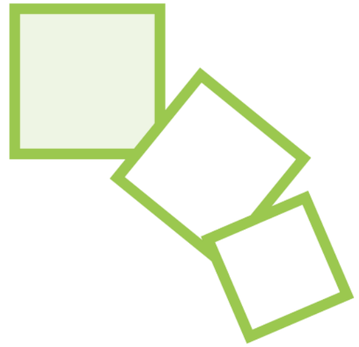


References Image

Workload manifest references the desired container image.

Workload Attributes

Defined in the YAML manifest(s) as configuration items.



Mutable

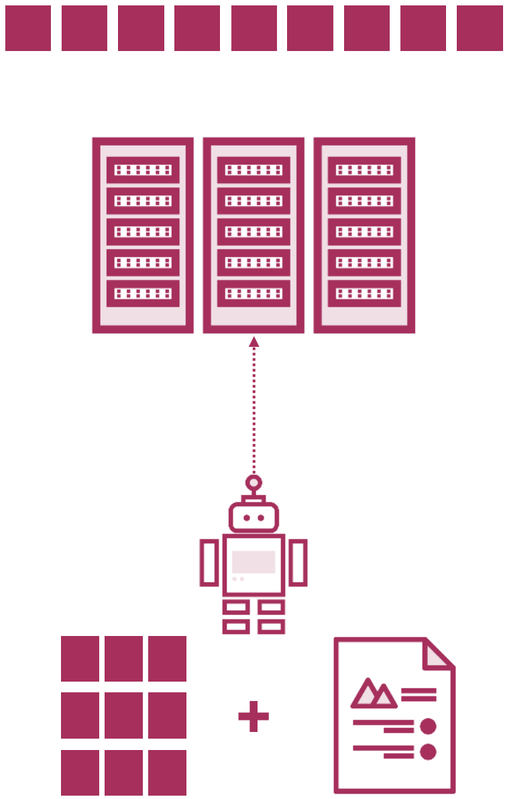
Can be altered after deployment using imperative commands.

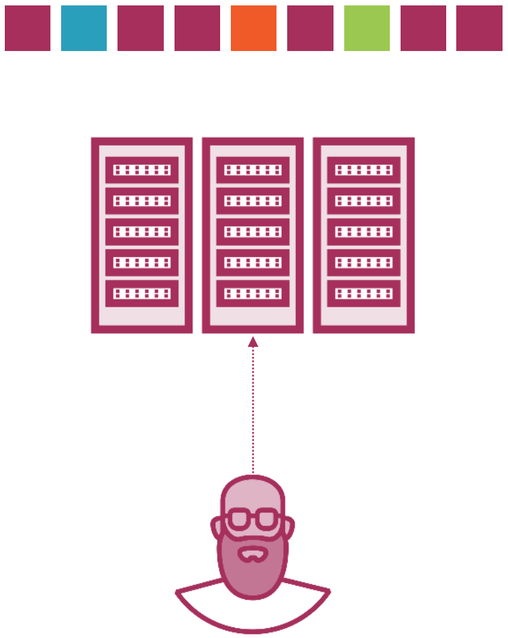
```
$ kubectl scale deployment myapp --replicas=5
deployment.apps/myapp scaled
```

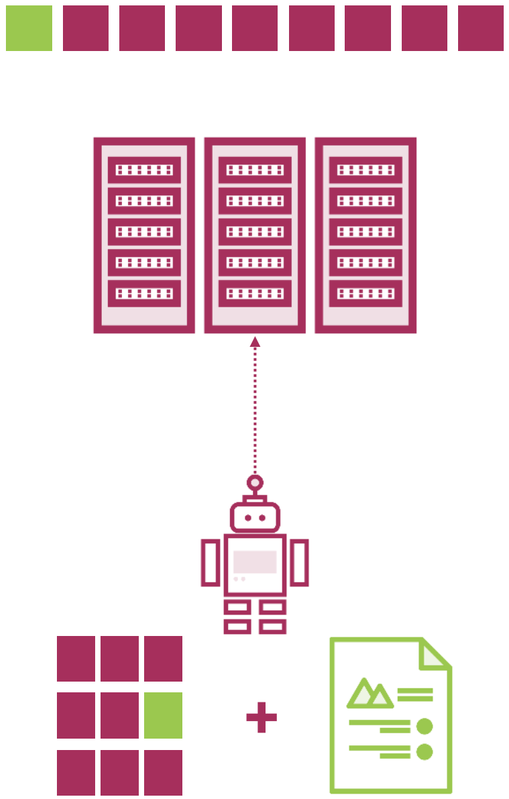
Imperative Commands

Can result in a mismatch between actual state, and the state represented by manifests

Configuration Drift







Managing the integrity of configuration in Kubernetes is essential for success

Up Next:
Using the GitOps Approach for Automating Deployments

Module Summary

What we covered:

- Application delivery using cloud native workflows
- Desired state, and desired state reconciliation
- Maintaining configuration integrity is an enduring problem

We need to know how to manage the configuration problem effectively!