The Kubernetes Saga: Secure GitOps & Infrastructure

Master the ways of Infrastructure as Code and GitOps - powerful tools for the modern DevOps Jedi.

Training path

Declarative vs imperative approaches

Different paths to infrastructure management

Infrastructure as Code

Mastering Bicep for resource orchestration

Secrets management

Protecting your sensitive data

git GitOps with Argo CD

Continuous deployment through Git

Declarative vs Imperative: Choose your Path

Declarative approach

Specify desired end state

System determines how to achieve it

Like Jedi mind influence

Examples: Terraform, Kubernetes manifests

Imperative approach

Define exact steps to execute

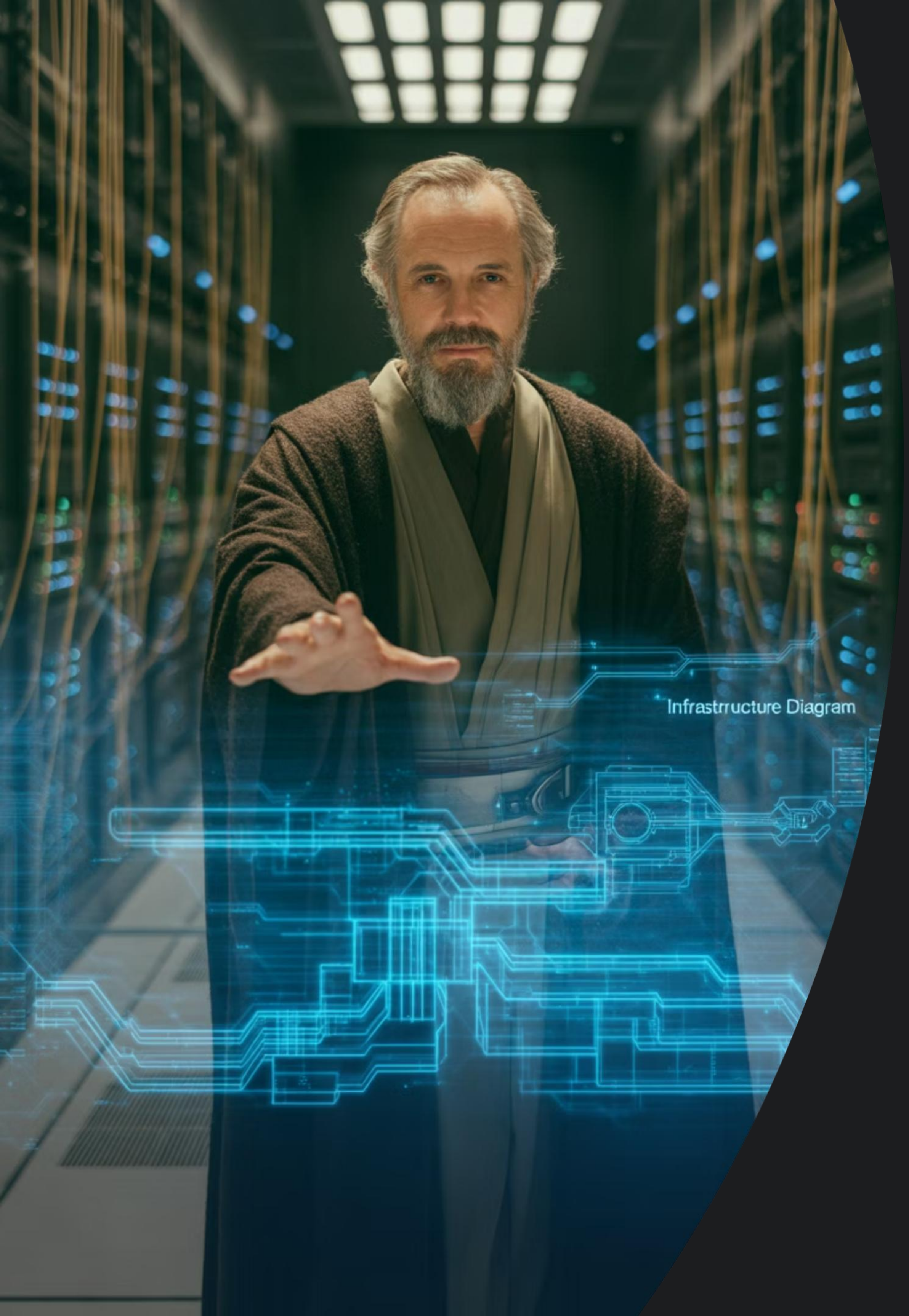
Specify how to achieve goal

Like direct Force commands

Examples: Shell scripts, manual CLI

Choose your Mastery

Both paths lead to Imperial dominance - declarative for strategic vision, imperative for tactical precision. Master Jedi combine both approaches as situations demand.



The Jedi mind trick: Declarative advantages

Idempotency

Run multiple times, same result

Safe to reapply without side effects

Self-healing

System corrects drift automatically

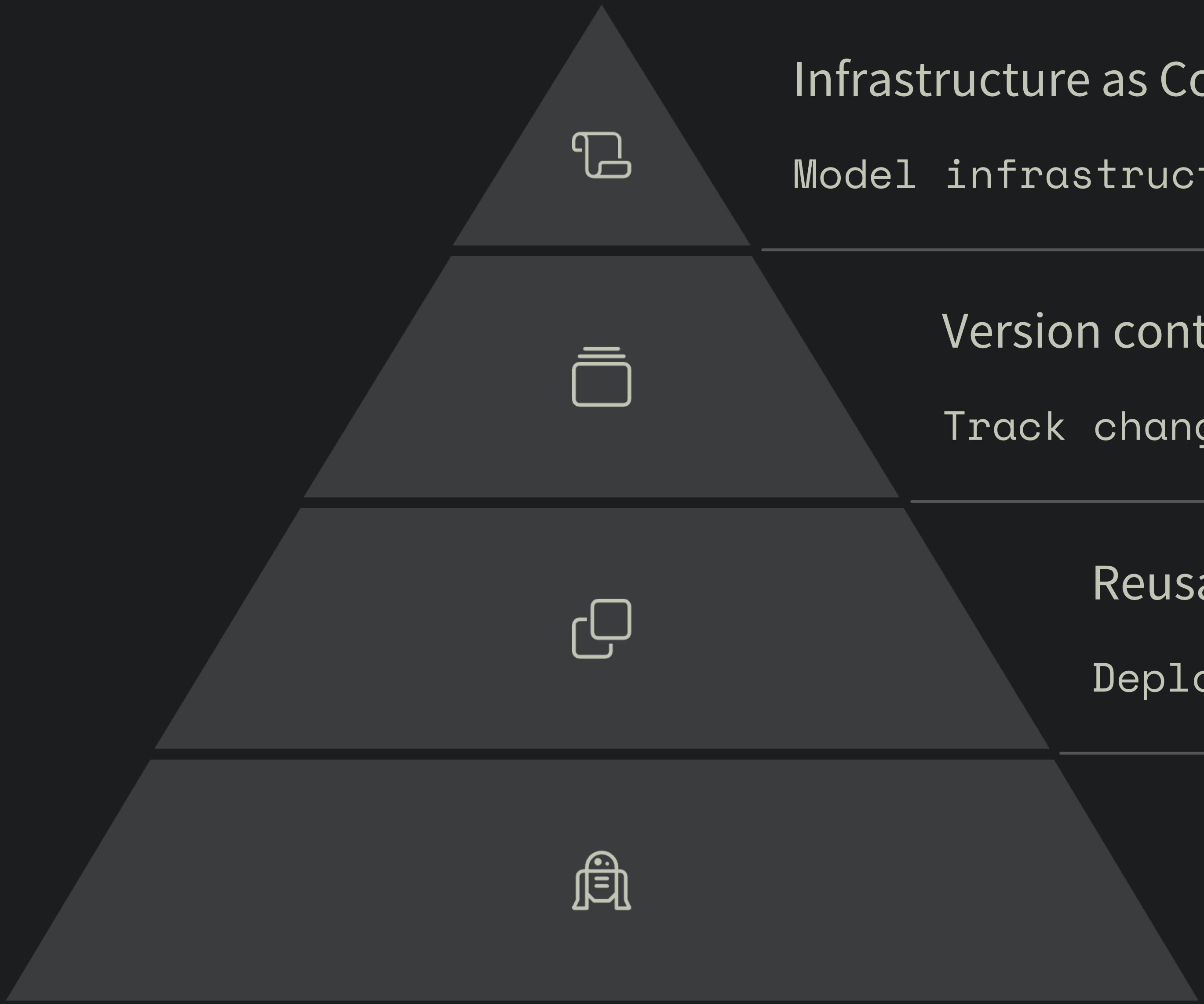
Ensures desired state maintenance

Documentation

Config files serve as living docs

Infrastructure visible in code

Infrastructure as Code: Building your Galaxy



Infrastructure as Code

Model infrastructure in text files

Version control

Track changes over time

Reusability

Deploy consistent environments

Automation

Reduce manual operations

Bicep: The elegant weapon

Domain-specific language

Designed for Azure deployments

Cleaner syntax than ARM templates

Modularity

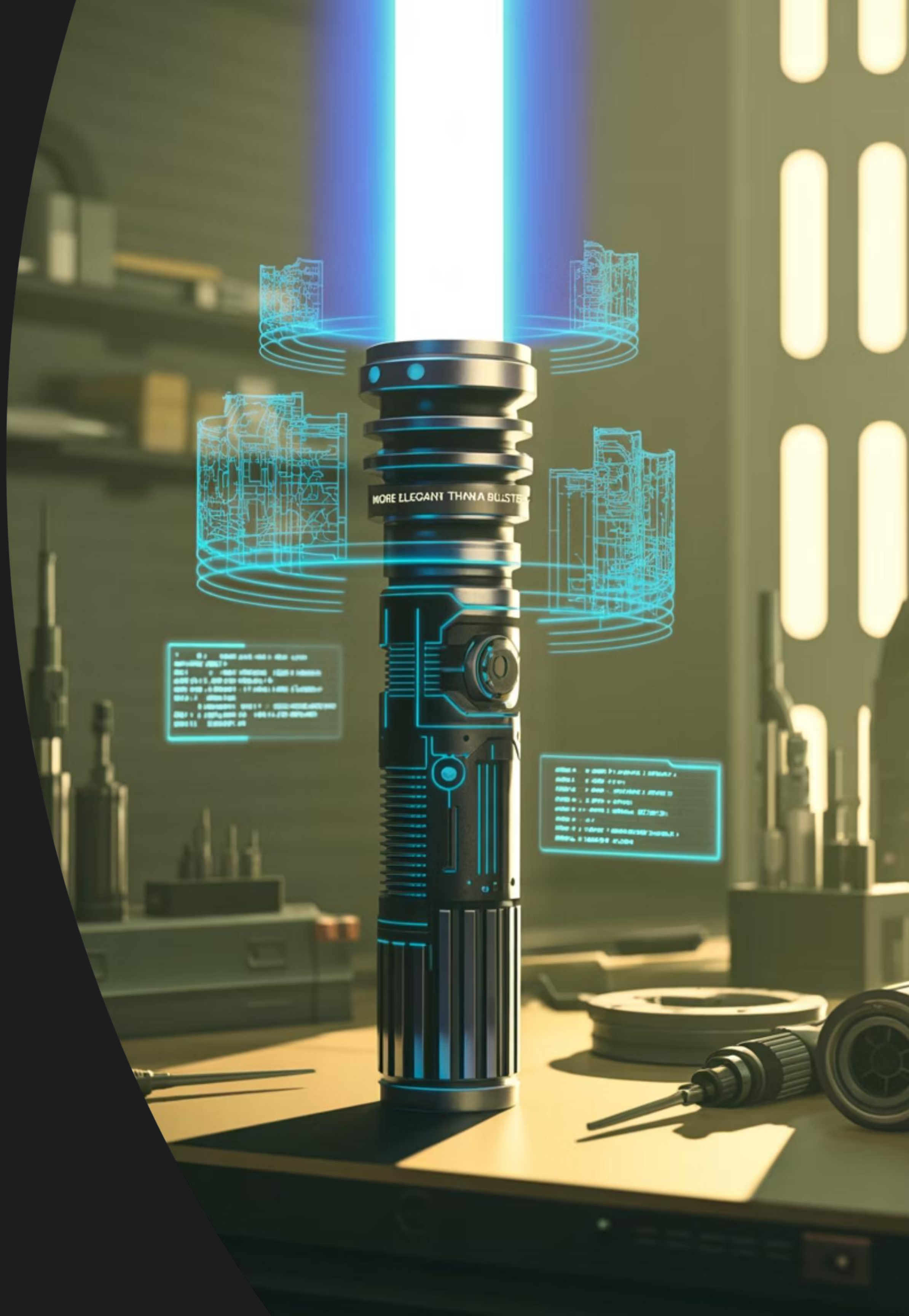
Reusable components

Compose complex infrastructure

Strong tooling

IntelliSense support

Type validation and linting



Bicep in action: Crafting your Lightsaber

```
resource aksCluster 'Microsoft.ContainerService/managedClusters@2025-
03-01' = {
  name: 'jedi-cluster'
  location: location
  properties: {
    kubernetesVersion: '1.32.3'
    enableRBAC: true
    agentPoolProfiles: [
      {
        name: 'agentpool'
        count: 3
        vmSize: 'Standard_D4ads_v6'
      }
    ]
  }
}
```

Lab 3.01 - Imperial Archives - IaC with Bicep

Establish secure archives

Create a dedicated Imperial Key Vault resource group and deploy a globally unique vault using Bicep templates for storing classified Imperial secrets.

Implement access controls

Assign proper RBAC permissions with the Key Vault Administrator role to prevent unauthorized access to sensitive Imperial intelligence.

Deploy advanced defenses

Enable Advanced Container Networking Services (ACNS) and L7 Network Policies to establish impenetrable security boundaries around cluster resources.

> "Without control, there is chaos. With Bicep, there is order." – Grand Moff Tarkin

Kustomize: The Force modifier

Base templates

Define standard Imperial configurations as base YAML manifests - the foundation of your Kubernetes deployments

Overlay patterns

Create environmental variations without duplicating code - apply targeted modifications for each star system

Patch deployment

Surgical precision with strategic, partial updates - maintain consistency while adapting to local planetary needs

Unlike the rigid Imperial protocols of Helm, Kustomize embraces a more Jedi-like flexibility. It requires no templates, no special syntax - just pure, declarative YAML that can be transformed through the subtle manipulation of the Force.

Lab 3.02 – Modularizing the Fleet with Kustomize

Base configuration

Create standard Imperial fleet manifests in **tie-squadron/base/** that define core deployments and services

Environmental overlays

Develop specialized configurations for dev and acc without duplicating code

Deploy with precision

Use **kubectl apply -k** to selectively deploy the right configuration to each environment

> "You don't deploy the fleet you want. You deploy the fleet your environment demands." – Admiral Thrawn

In this Imperial exercise, you'll utilize Kustomize to standardize deployments across the galaxy – from the frozen outposts of Hoth to the scorching factories of Mustafar – ensuring configuration consistency while adapting to local planetary requirements.

Helm: The Jedi package manager

Unified Charts

Package Kubernetes manifests into reusable charts, like holocrons containing operational knowledge.

Hyperspace Deployment

Install complex applications with a single command. Deploy entire fleets as easily as single starfighters.

Release History

Track deployments and rollback to previous versions faster than the Millennium Falcon's escape maneuvers.

Helm acts as your astromech companion, navigating the complex galaxy of Kubernetes deployments with precision and reliability.

Hyperspace Commands: Navigating Helm

Search the Galaxy

```
helm search repo
```

Scan the available chart repositories like scouting for Imperial outposts.

Download Holocrons

```
helm pull
```

Extract chart archives locally before deployment, like studying ancient Jedi texts.

Deploy the Fleet

```
helm install rebellion ./chart
```

Launch applications into the cluster with precise coordinates and configurations.

Tactical Retreat

```
helm rollback rebellion 1
```

Jump back to previous versions faster than lightspeed when confronting Dark Side bugs.

Like an astromech droid's tools, these commands give you complete control over your Kubernetes deployments across the galaxy.

Lab 3.03 - Imperial Deployment Protocol: Helm Charts

Deploy Imperial Fleet

Launch applications using public Helm charts from the galactic repository.

```
helm install
```

Monitor your forces

Inspect deployed releases and verify configurations across star systems.

```
helm list
```

Customize Imperial standards

Deploy with custom values.yaml for precise resource allocation and configuration.

> "You may fire when ready... just make sure you installed it with the correct values." – Grand Moff Tarkin

Secrets management: Guarding the Holocron

Identify secrets

API keys, passwords, certificates

Store securely

Use dedicated secret stores, not Git

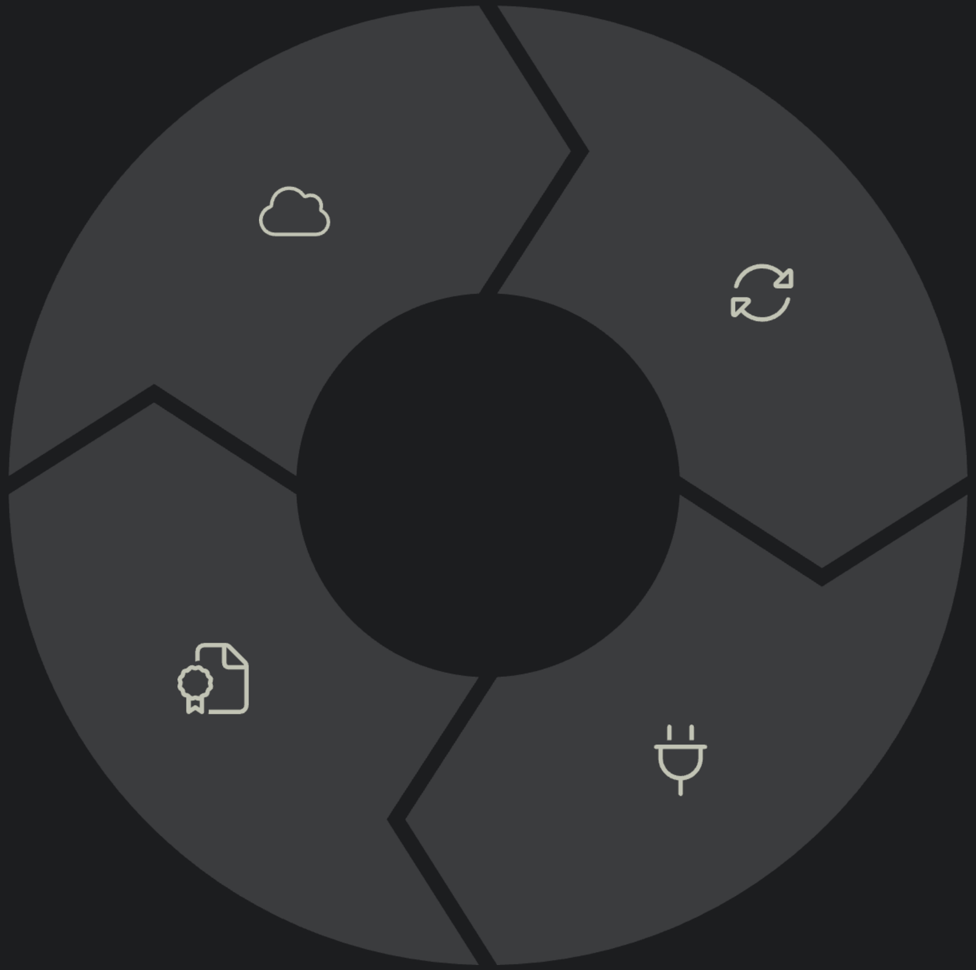
Deploy with External Secrets Operator

Fetch from vault to Kubernetes

Rotate regularly

Automate credential renewal

External Secrets Operator: The Force connection



External vault

Azure KeyVault, AWS Secrets Manager, HashiCorp Vault

Secret operator

Runs in cluster, connects to external provider

ExternalSecret

Maps external secrets to K8s Secret objects

(Cluster)SecretStore

Defines connection to external system

Lab 3.04 - Secrets from the Shadows - External Secrets with Imperial Archives

1. Configure Workload Identity

   Establish secure federation between service accounts and Azure without static credentials.

2. Provision Azure resources

   Create Managed Identity using Bicep to securely access to Imperial Archives (Key Vault).

3. Deploy External Secrets Operator

   Install the operator with Helm to establish communication with external vaults.

4. Connect to Imperial Archives

   Create ClusterSecretStore and ExternalSecret to retrieve classified information securely.

> "Blueprints fade, keys get stolen... only automated policy persists." - Moff Gideon, Cybersecurity Division

Cluster bootstrapping: The initial spark

Define infrastructure

Bicep templates for AKS cluster

Core add-ons

Gateway controller, observability, security management

Identity management

Service accounts, RBAC setup

GitOps tooling

Install Argo CD as bootstrap application

GitOps: The way of the Force

Git as single source of truth

All desired state stored in repositories

Pull-based deployment

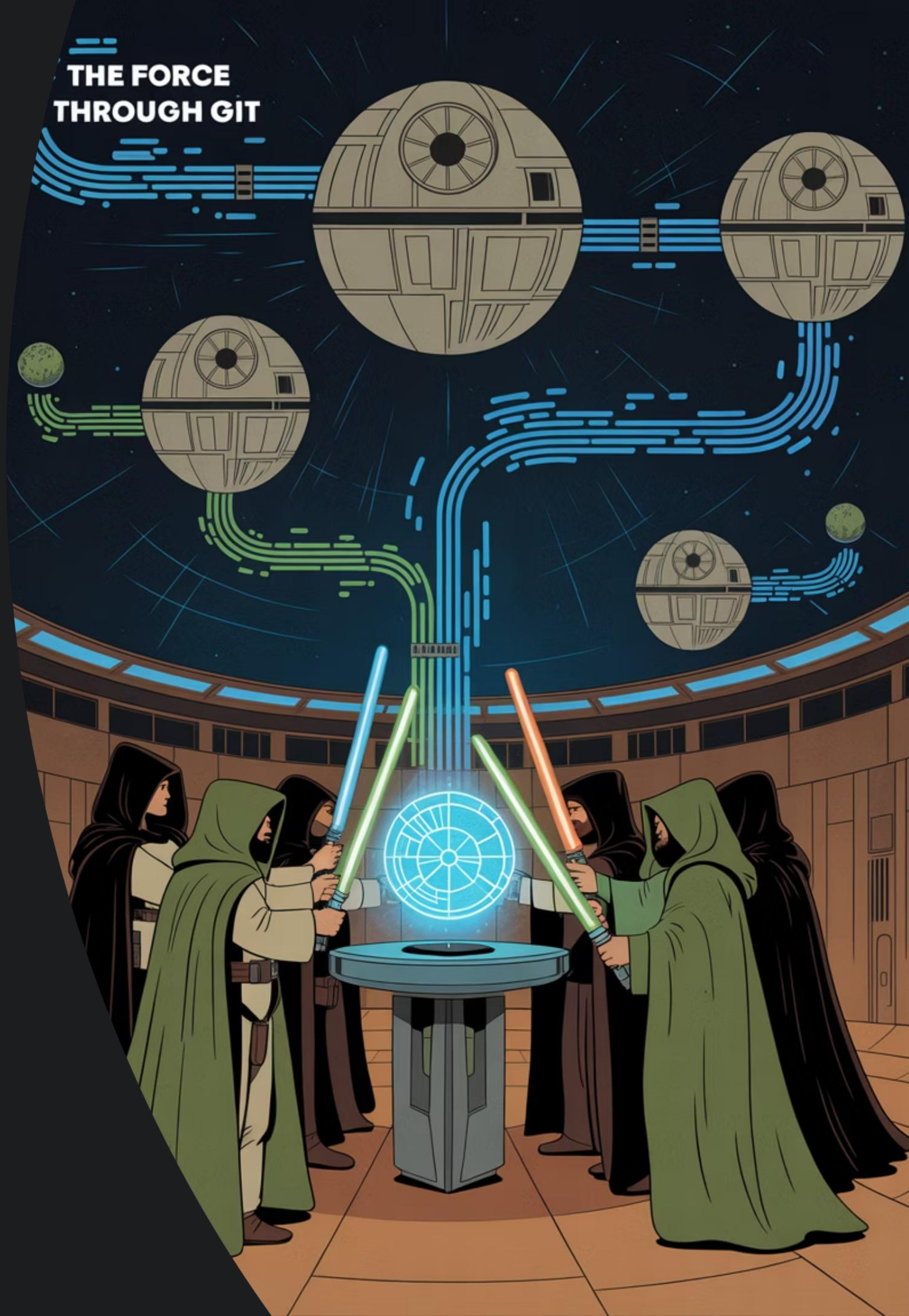
Agents pull approved changes from Git

Detect & resolve drift

Automatic reconciliation with desired state

Observable & auditable

Every change tracked and reviewable



Argo CD: Your GitOps astromech

Kubernetes-Native

Runs inside your cluster

UI Dashboard

Visual deployment status tracking

Automated Sync

Keeps cluster in sync with git

GitOps implementation: The Jedi architecture

Store manifests in git

Kubernetes YAML, Helm charts, Kustomize

Everything must be declarative

Install Argo CD

Deploy controller into cluster

Configure Git repository access

Define applications

Point to Git sources

Set sync policies and targets

Automate workflow

Git PR reviews become deployment approvals

Merge to trigger deployment

Argo CD Applications: Deploying the Fleet

Argo CD manages different Kubernetes resource types that handle networking, routing, storage, and configuration needs for your applications.

Service

Exposes applications on a network.

HTTPRoute

Routing rules for HTTP/S traffic.

ConfigMap

Holds non-sensitive configuration data.

Deployment

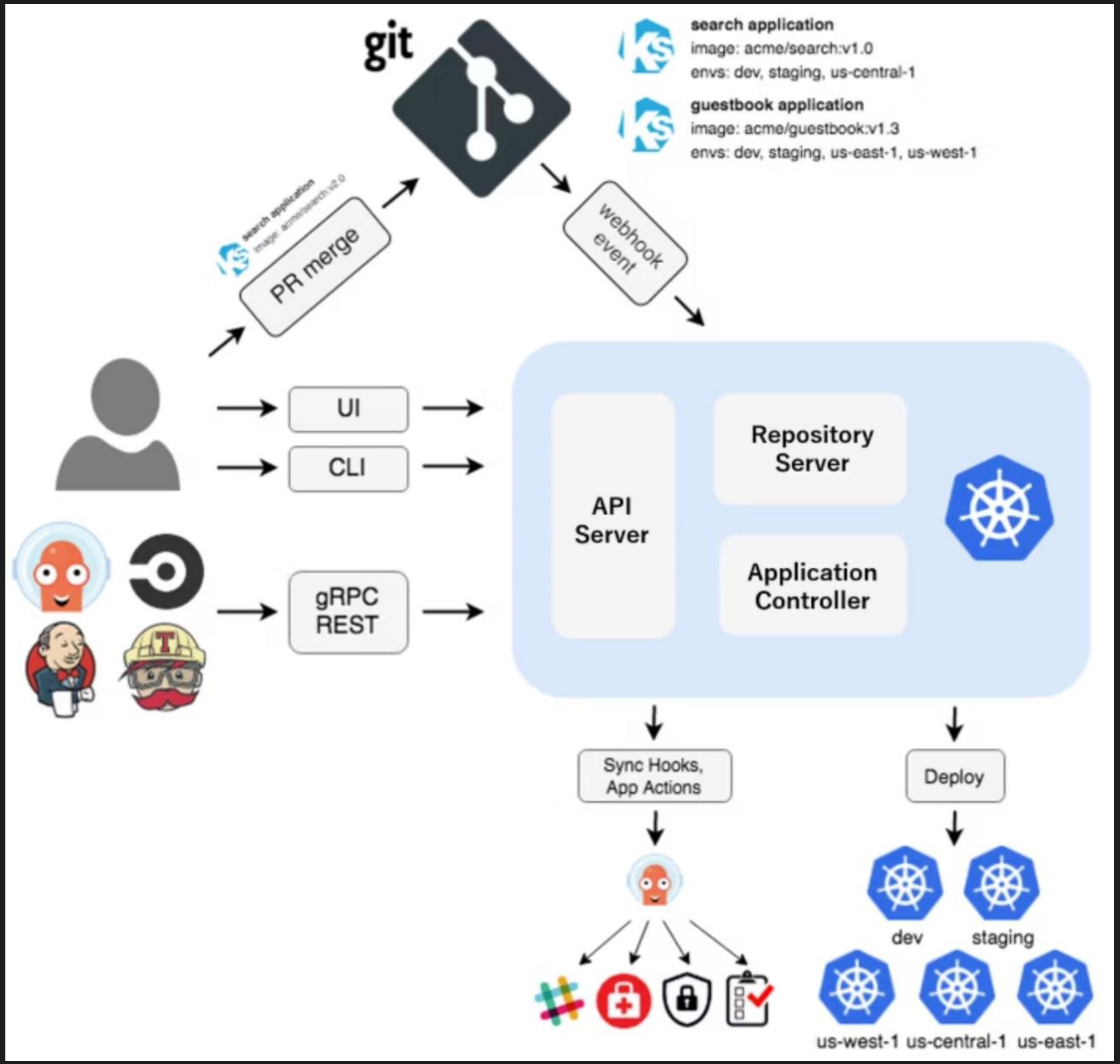
Application

ExternalSecret

Securely stores sensitive information like database credentials and API keys.

PersistentVolumeClaim

Requests storage resources for pods based on size and access requirements.


git
search application
image: acme/search:v1.0
envs: dev, staging, us-central-1
guestbook application
image: acme/guestbook:v1.3
envs: dev, staging, us-east-1, us-west-1
search application
image: acme/search:v2.0
PR merge
webhook event
UI
CLI
gRPC REST
API Server
Repository Server
Application Controller
Sync Hooks, App Actions
Deploy
dev
staging
us-west-1
us-central-1
us-east-1

Lab 3.05 - Initiating Fleetwide Autopilot - Installing Argo CD

Install Argo CD Controller

Deploy using Helm with system node tolerations and insecure mode.

```
helm install argocd
```

Verify Deployment

Check pod readiness and access the dashboard.

```
kubectl get pods -n argocd

kubectl port-forward
svc/argocd-server -n argocd
8080:443
```

Access Control Center

Navigate to the Argo CD UI at http://localhost:8080.

> Let the fleet deploy itself. Your job is to command, not babysit pods. - Grand Admiral Thrawn

Monorepo vs multi-repo: Choosing our Path

Monorepo approach

Single repository containing all code, configurations, and infrastructure definitions in one centralized location.

Multi-repo approach

Distributed repositories organized by component, service, or team with separated concerns.

Our choice: Monorepo

We've chosen the monorepo approach for simplicity, visibility, and streamlined workflows.

Monorepo advantages

- Simplified dependency management
- Atomic changes across multiple components
- Unified version control and history

Multi-repo challenges

- Increased complexity in repository management
- Potential for version incompatibilities
- Requires robust CI/CD pipeline coordination

|-- clusters | |-- dta | | |-- apps | | |-- platform | | `-- root-app | `-- prd | |-- apps | |-- platform | `-- root-app

The Galaxy Repository: Our monorepo structure

clusters

The command center housing all cluster configurations.

environments

DTA for development/testing and PRD for production deployments.

components

Apps, platform services, and root-app orchestration per environment.

```
|-- clusters
|   |-- dta
|   |   |-- apps
|   |   |-- platform
|   |   `-- root-app
|   `-- prd
|       |-- apps
|       |-- platform
|       `-- root-app
```

Our Imperial repository uses consistent structure across environments. The root-app functions as the primary Argo CD application, orchestrating all deployments within each cluster.

Platform components deploy first, establishing core infrastructure before applications launch.

Multi-environment: Across the Galaxy

Helm approach

Environment-specific values stored in dedicated directories.

Chart Definition

Chart.yaml defines app metadata and dependencies.

Environment Values

Values.yaml files customized per environment in /environments.

Templates

Kubernetes manifests with variables for environment injection.

Kustomize approach

Base configurations with environment-specific overlays.

Base Resources

Core manifests defined once in /base directory.

Overlays

Environment-specific patches in /overlays/{env} directories.

Deployment

ArgoCD applies the appropriate overlay per environment.

> Navigate the stars with confidence. Each environment is a different planet, but the navigation systems remain the same.

Multi-environment: Across the Galaxy

Development environment

Fast iteration, lenient policies

Auto-sync, branch-based deployments

Staging environment

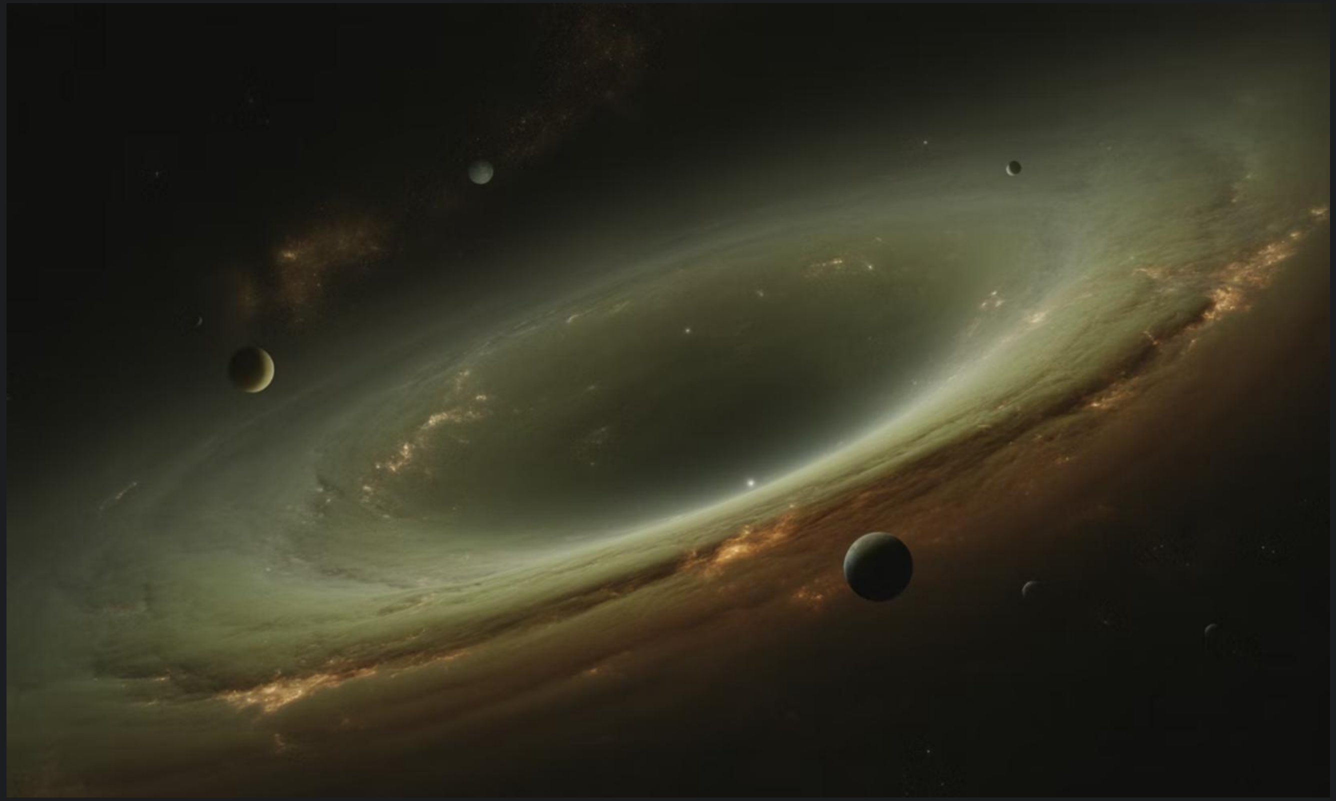
Production-like for testing

Manual promotion, automated testing

Production environment

Stable, controlled changes

Manual approval, stricter policies



Sync policies: Controlling the Force

Manual sync	Changes applied only when requested
Automated sync	Changes applied automatically when detected
Self-healing	Revert unauthorized cluster changes
Prune resources	Remove resources deleted from Git
Apply only	Never delete resources, only create/update

Lab 3.06: Automating Imperial Fleet

GitOps and Kustomize

Create repository structure

Organize manifests in Git with proper hierarchy for Imperial Fleet deployments.

Define ArgoCD Application

Create dev-application.yaml to declare the GitOps deployment configuration.

Specify source repository, target cluster, and sync policies.

Apply and monitor

Deploy the Application to ArgoCD and observe auto-synchronization.

Watch as squadron and service configurations automatically deploy from Git.

> "Victory is achieved not through chaos, but through precision and automation." – Grand Admiral Thrawn

Lab 3.07 – Automating Rebel Fleet GitOps and Helm

Create repository structure

Organize your GitOps repository with proper directories for the rebel-fleet and x-wing squadron configuration.

Define ArgoCD Application

Create dev-application.yaml that references the Bitnami NGINX Helm chart and your custom dev-values.yaml.

Configure target namespace "rebel-fleet" and appropriate sync policies.

Sync and verify

Commit changes, trigger sync in ArgoCD UI, and verify deployment success:

Argo CD projects: Securing the Galaxy

Namespace isolation

Restrict deployments to specific K8s namespaces

Source repositories

Limit which Git repos can be deployed

Destination clusters

Control which clusters receive deployments

Resource kinds

Allow only specific Kubernetes resource types

Sync schedules

Define when applications are allowed to sync

RBAC: Council permissions

Roles

Admin, Developer, Viewer

Projects

Logical grouping of applications

Authentication

local users, Microsoft Entra ID, OIDC

Policies

Define allowed actions per role

Lab 3.08 – ArgoCD Projects: Jedi vs. Sith Sector isolation

Create projects

Define separate ArgoCD Projects for Jedi and Sith factions with strict boundaries

Configure boundaries

Set source repositories, destination namespaces, and resource permissions

Define sync windows

Implement time-based deployment controls: Sith (8AM-8PM) vs Jedi (8PM-8AM)

Update applications

Modify tie-squadron and x-wing-squadron to use their respective projects

> "A Jedi does not deploy into Sith territory... unless he's debugging." – Master Windu

This lab implements faction isolation using ArgoCD Projects as security boundaries, preventing cross-deployment while enabling controlled version updates and scheduled deployment windows.

May the GitOps Force be with you

Declarative infrastructure

Specify what, not how

Infrastructure becomes predictable

Git-driven operations

PRs replace manual changes

History tracked for compliance

Automated reconciliation

Self-healing systems

Drift automatically corrected

Bonus Lab 3.09 - Imperial Troubleshooting Protocol

Even the most perfectly designed systems require diagnostics when the unexpected occurs. The Imperial fleet needs officers capable of resolving GitOps failures.

1 The Silence of Sector 7

Debug a service that can't route traffic despite running pods.

- Find missing endpoints
- Correct service selectors

2 The Vault Whispered Nothing

Resolve ExternalSecret failures preventing TLS certificate sync.

- Fix SecretStore configuration
- Correct Azure Key Vault references

3 Echoes in the Void

Troubleshoot persistent volume mounting issues in PostgreSQL deployment.

- Align PVC names
- Fix storage configuration

4 Silent Credentials

Diagnose RBAC permissions preventing secret access.

- Verify ServiceAccount bindings
- Correct resource access roles

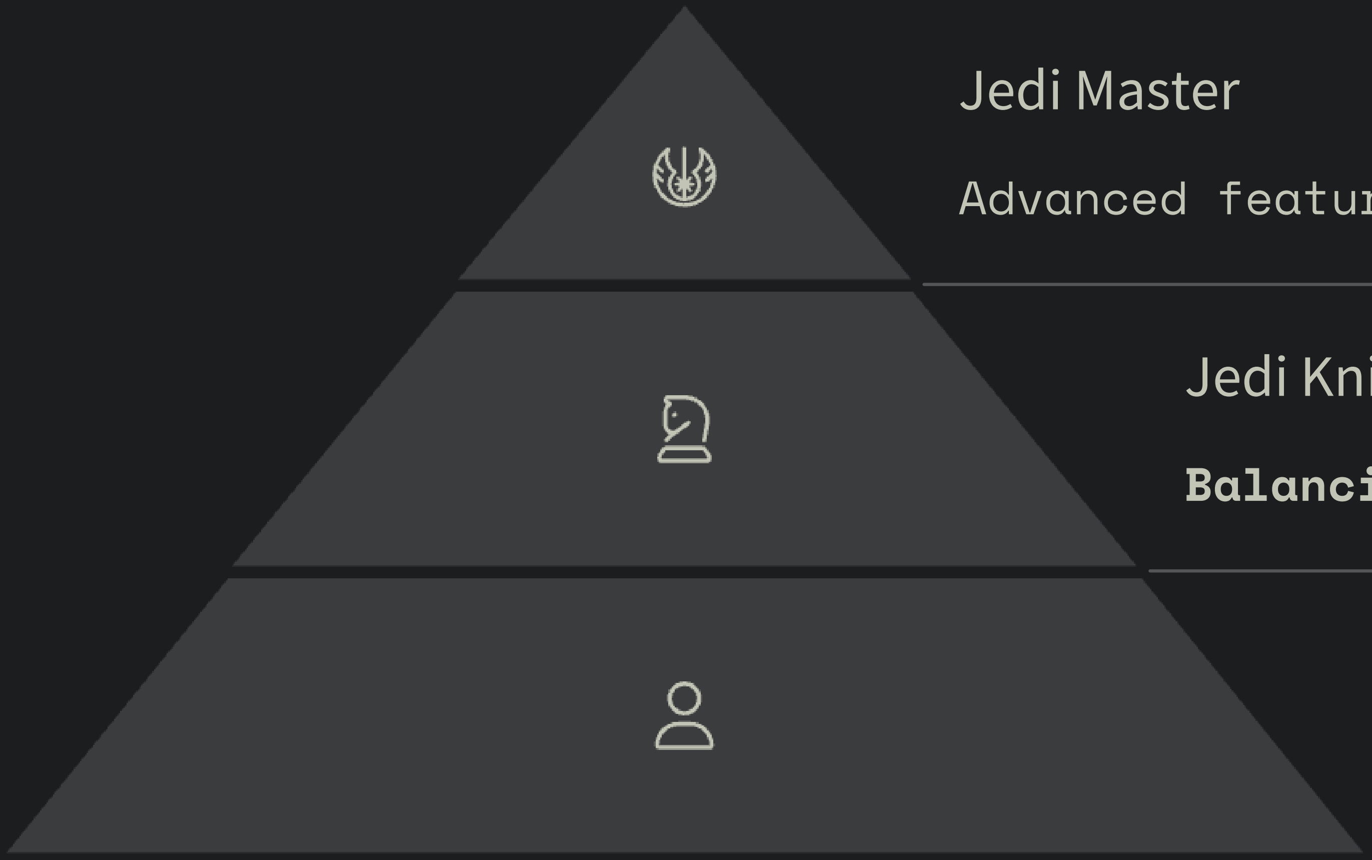
5 The Phantom Startup

Fix multi-container pod initialization sequence.

- Debug init container failures
- Resolve shared volume conflicts

> "In the void of space, a broken signal can doom an entire fleet. Debug methodically, fix decisively." — Grand Moff Tarkin

Rise through the Ranks



Jedi Master

Advanced features and customizations

Jedi Knight

Balancing wisdom and action - you're battle-tested!

Padawan

Basic concepts mastered