GitOps From Commit to Deploy

Daniel Hinojosa

GitOps Continuous Deployment

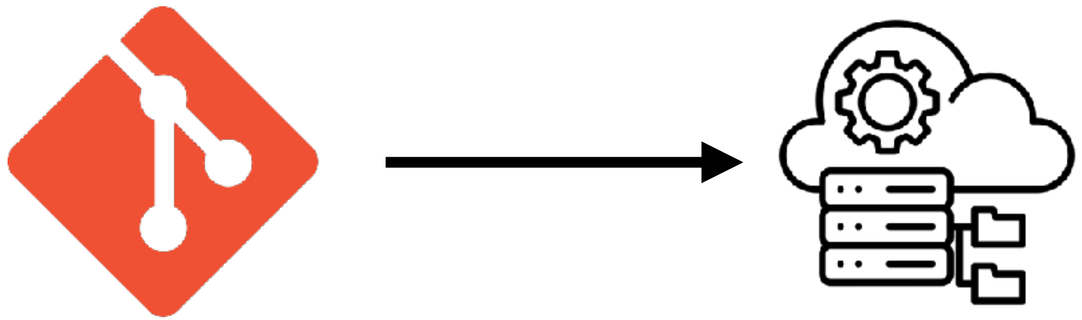
In this Presentation

- GitOps
- What is Argo CD?
- Installing Argo CD
- Syncing Applications
- Pruning Applications
- Argo Events, Workflows
- Argo Rollouts & Canaries

GitOps

Infrastructure as Code (IaC)

- Practice of managing and provisioning infrastructure through code instead of manual processes
- All infrastructure is defined in a repository and run from the repository
- Typically done with Ansible, Terraform, Kubernetes manifests, but can also be done with scripts



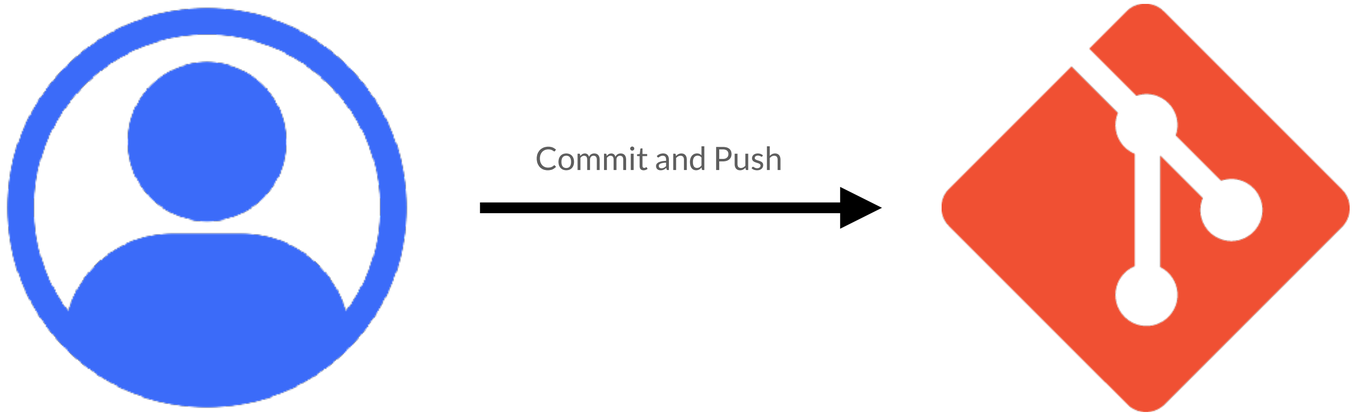
More than just Infrastructure

- Infrastructure as Code
- Networking as Code
- Security as Code
- Configuration as Code

EVERYTHING IS TRACKED,
EVERYTHING HAS A HISTORY,
EVERYTHING HAS A DEVELOPER

Git but not IaC

You’re using Git, but is that enough?



IS THAT ALL?

Benefits of a Git Repo

- Single Source of Truth - All IaC files are located in a Git repository
- Everyone has access to it

What else do we need though?

- Pull Requests
- Code Reviews
- Automated Tests
- Automated Linters
- Automated Deployments

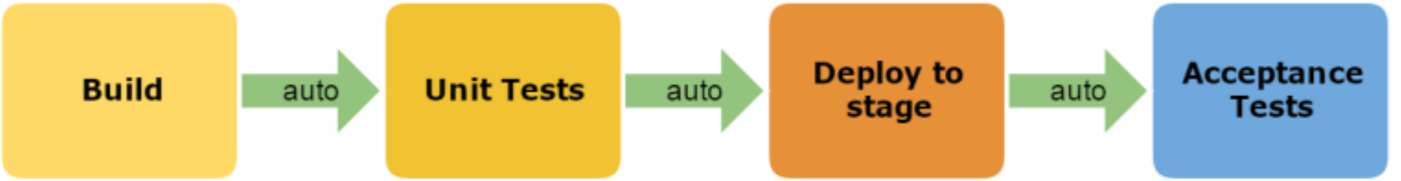
Separate Repositories

Why maintaining separate repositories is essential

Separate Repositories

- **For GitOps you separate the application repository from the configuration repository**
  - **Application Repository** contains source code, build scripts, and tests for the application.
  - **Configuration Repository** stores Kubernetes manifests, Helm charts, or Kustomize configurations that define the desired state of the application in different environments (e.g., staging, production).
- Reasons to do this are separation of concerns: application and deployment
- Team autonomy: Application development and Configuration deployments are performed by different teams
- *Avoids having to redeploy application because of configuration changes and vice versa*

Continuous Integration



Continuous Delivery

Continuous Deployment

Continuous Integration

Continuous Integration

- Practice where developers integrate code changes into a shared repository frequently, ideally multiple times a day.
- Each integration triggers an automated build and test process, ensuring that the new changes do not break the existing codebase.
- In GitOps,
  - Ensures the integrity of application manifests and Kubernetes configurations before they are deployed.
  - Automatically builds and pushes container images to a registry after successful testing.
  - Validates that declarative configurations match the intended state of the system.
  - Scanning Containers for Vulnerabilities
  - More...

Lab: Continuous Integration

- Let’s fork our repository so that we all have the project
- Let’s review the Github Actions and see how it works
- Let’s review the Security aspects, especially the secrets handling

Continuous Delivery

Continuous Delivery

- Automates the deployment process up to the production environment but requires manual approval or triggers for the final release.
- Allows teams to decide when to deploy based on business needs, schedules, or additional verification.
- Ensures the system is always in a deployable state, even if deployments aren’t immediate.
- Offers a balance between automation and control, reducing risks in complex or sensitive environments.
- Use Cases: Suitable for organizations with compliance requirements, controlled release cycles, or critical production environments.

Continuous Deployment

Continuous Deployment

- Fully automated process of releasing every change that passes CI testing to production without manual intervention.
- Removes human decision-making from deployment, relying entirely on automated testing and pipelines.
- Enables rapid delivery of new features, bug fixes, and updates to end-users.
- Requires a high level of confidence in automated testing and monitoring to avoid issues in production.

Argo CD

Argo CD

- A declarative, GitOps-based continuous delivery tool for **Kubernetes**.
- Features:
  - Synchronizes Kubernetes clusters with Git repositories.
  - Supports declarative configuration using tools like **Kustomize** and **Helm**.
  - Provides real-time monitoring of application state.
  - Enables automated and manual rollbacks.

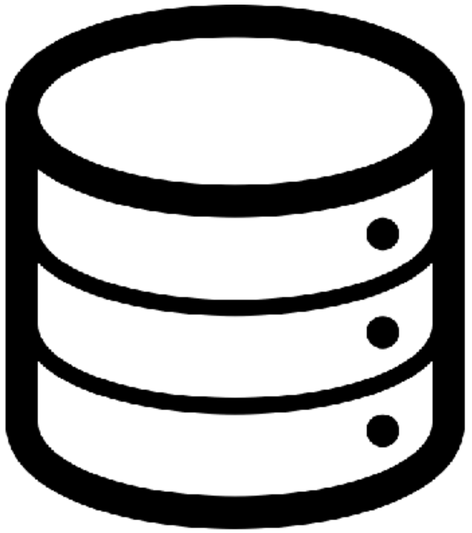
Components of Argo CD

- **API Server:** Manages interactions with the Argo CD system.
- **Controller:** Monitors and applies the desired state to the cluster.
- **Web UI/CLI:** Provides user interfaces for managing applications.

Deploy Infrastructure First

Setting up your Infrastructure First

- Setup Databases, Pub Subs, and Message Queues first
- Frequent changes to these critical components can introduce instability.
- Ensures infrastructure is thoroughly tested and stable before applications are deployed.
- Infrastructure changes often involve schema updates, data migrations, or cluster reconfigurations.
- Including these in every deployment pipeline increases the complexity of CI/CD workflows.



Created by Shmidt Sergey
from Noun Project

Installing Argo CD

Installation

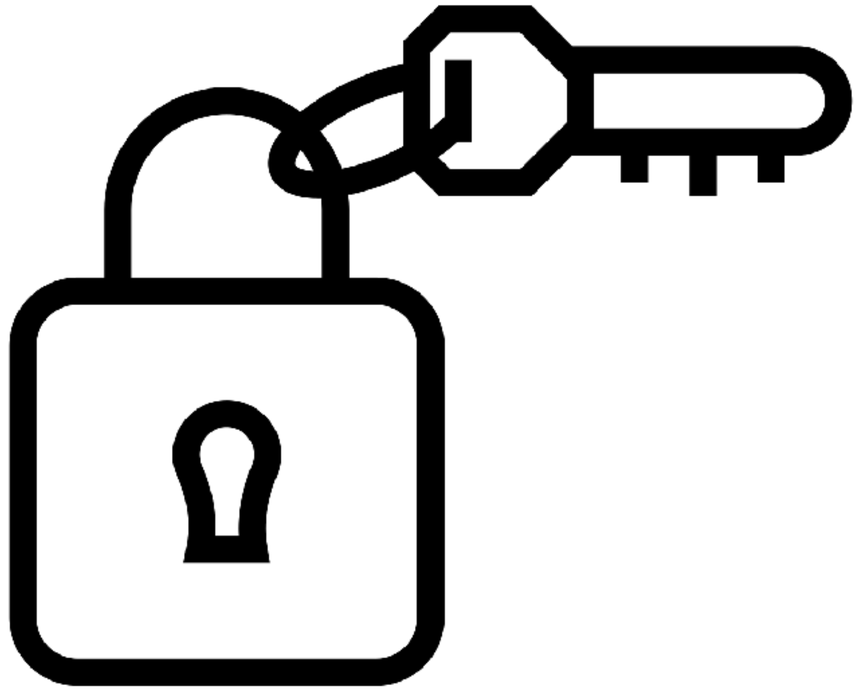
- Kubernetes cluster (v1.21 or later recommended).
- `kubectl` installed and configured to access the Kubernetes cluster.
- Sufficient permissions to deploy resources (e.g., Cluster Admin).

```
$ kubectl create namespace argocd
$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

- You can then on local development, access your argocd via port-forwarding
- Port forwarding does not involve TLS so that would either require making ArgoCD insecure via setting, or you would override it on the browser

Setting up Ingress/Production

- Since typically this is installed on a Kubernetes and it will be used by others, this will be exposed through an ingress and ingress controller
- Highly recommended to serve Argo CD using TLS and a domain name.
- ArgoCD has also a CLI that you can use to interact with the web application, and that too will require secure connections.



Lab: Show the Installation

- Let’s show the installation of ArgoCD

Using Argo CD

Using Argo CD

- Argo CD can be operated on either by Web UI or CLI
- UI:
  - Ideal for quick operations, visual monitoring, and debugging.
  - Suitable for non-technical users or those new to Argo CD.
- CLI:
  - Best for automation, CI/CD pipelines, and scripting repetitive tasks.
  - Useful for advanced users who need fine-grained control or want to integrate Argo CD with other tools

Retrieval of the admin password

First, in both the CLI, and the UI, you will need to get the initial password from a Kubernetes secret, it is of course recommended that change it after the first use.

```
$ kubectl get secret -n argocd argocd-initial-admin-secret -o jsonpath="{.data.password}"
| base64 -d
```

Getting started with the CLI

Logging in, into Argo CD

- Logging into ArgoCD from the CLI
  - Must be done securely and through TLS
  - Can be done by providing the URL

```
$ argocd login argocd.tiered-planet.net --skip-test-tls --grpc-web

tGB8G9IhLN4lK2TR
```

Registering your clusters

- Let’s say we have two clusters: staging and production
  - We will use the contexts
  - Create a namespace argocd on both
  - Add the cluster to ArgoCD so that it knows about them

```
$ kubectl config use-context staging-cluster-1
$ kubectl create namespace argocd
$ argocd cluster add staging-cluster-1 --system-namespace argocd
```

```
$ kubectl config use-context prod-cluster-1
$ kubectl create namespace argocd
$ argocd cluster add prod-cluster-1 --system-namespace argocd
```

Create the necessary namespaces

Since we are creating applications in the a particular `namespace` we can create that namespace in each of the clusters

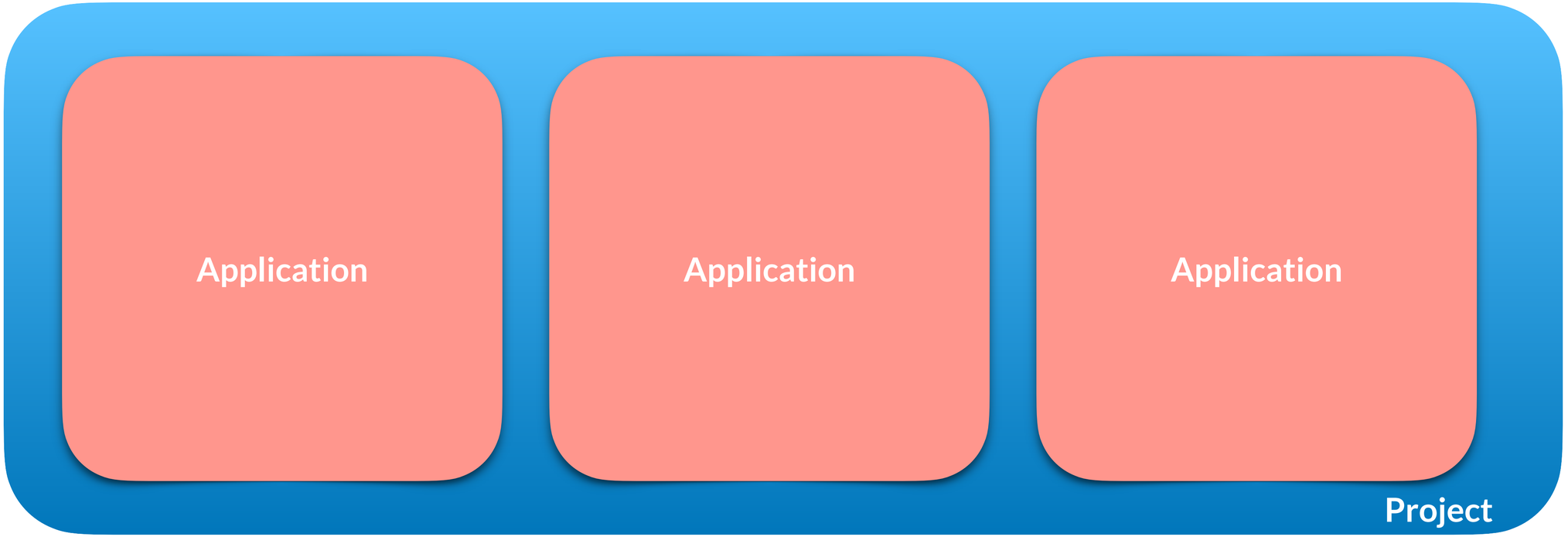
```
$ kubectl config use-context staging-cluster-1
$ kubectl create namespace gitops-todo
```

```
$ kubectl config use-context prod-cluster-1
$ kubectl create namespace gitops-todo
```

Creating a Project

Argo CD Project

- **A project** is a logical grouping or boundary for applications that allows you to define and enforce access controls, clusters, resource usage, and deployment rules.



Argo CD Project in UI

- On the left-hand menu, click **Settings**
- Under the settings page, select **Projects**
- Click on **New Project**
- Add Configurations
  - **Source Repositories:** Add the Git repository URLs or wildcards that this project can access.
  - **Destinations:** Specify the clusters and namespaces where applications in this project can deploy.
  - **Cluster Resource Access:** Define which cluster-wide Kubernetes resources are allowed or denied for this project.

Argo CD Project in CLI

- Creating a project via the CLI is very straightforward

```
$ argocd proj create <PROJECT_NAME>
```

- You can configure many project settings by the command line

```
$ argocd proj allow-cluster-resource PROJECT GROUP KIND [flags]
```

Creating an Application

Creating an Application on UI

Interacting on the Web

Creating an Application to Sync

- Click on “Applications” in the navigation bar.
- Click on the “New App” button in the top-right corner.
- Fill in Application Details, like Application Name and Project
- Fill in Source Configuration: Repository URL, Revision, Path
- Fill in Destination Configuration: Cluster where the application will be deployed (e.g., `https://kubernetes.default.svc` for in-cluster).
- Fill in Sync Policy:
  - Manual - Requires you to trigger synchronizations manually.
  - Automated - Automatically syncs changes from Git and optionally prunes orphaned resources.

Creating an Application in CLI

Interacting on the Command Line

Create an Application using CLI

- Creating an application can purely be done in the CLI by providing switches for everything required

```
$ argocd app create <app-name> --repo <repo-url> --path <path> --dest-server <cluster-
url> --dest-namespace <namespace>
```

- Here is an example of creating the application

```
$ argocd app create my-app \
  --repo https://github.com/example/repo.git \
  --path manifests \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace default
```

Create an Application using CLI and Manifests

- Creating an application can be done by a custom resource definition (CRD) and applied via manifest

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/example/repo.git
    targetRevision: HEAD
    path: manifests
  destination:
    server: https://kubernetes.default.svc
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Demo: Show an Application in the UI

- Let’s show an application from the UI and how to create one

Performing Deploys with Kustomize

What is Kustomize?

- A Kubernetes-native tool for managing and customizing configuration files.
- Allows overlays and transformations of YAML manifests without modifying the original files.

Features of Kustomize

- **Base and Overlays:**
  - Base: Common configurations shared across environments.
  - Overlays: Environment-specific configurations (e.g., staging, production).
- **Declarative Management:**
  - Uses `kustomization.yaml` to define transformations and resource references.
- **Built-in Patches:**
  - Supports strategic merge patches and JSON patches.
- **No Templates:**
  - Operates purely on YAML, avoiding the need for templating.

How is Kustomize used with Argo CD?

- **Integration:**
  - Argo CD natively supports Kustomize as an alternative to plain YAML manifests.
  - Automatically processes `kustomization.yaml` files in the repository.
- **Use Cases:**
  - Environment-Specific Deployments:
  - Define separate overlays for staging, production, or testing.
- **Customizing Shared Resources:**
  - Apply patches or set environment-specific variables.
- **Managing Application Lifecycles:**
  - Incrementally roll out updates with reusable configurations.

Lab: Kustomize

- Let’s show what Kustomize is in the configuration repository and how it is used within Argo
- We can install Kustomize and validate the yaml file

Performing Deploys with Helm

What is Helm?

- A package manager for Kubernetes that simplifies the deployment and management of applications using Helm charts.
- Helm charts are collections of YAML templates and configuration files that describe Kubernetes resources.

Features of Helm

- **Chart Repositories:**
  - Centralized locations to store and distribute application charts.
- **Templating Engine:**
  - Allows dynamic generation of Kubernetes manifests using variables.
- **Versioning:**
  - Supports version control for application releases.
- **Rollbacks:**
  - Simplifies rolling back to previous application versions.

How is Helm used with Argo CD?

- **Integration:**
  - Argo CD supports Helm natively, enabling the deployment of applications directly from Helm charts.
  - Works with both remote chart repositories and local charts stored in Git.
- **Use Cases:**
  - Deploying complex applications with extensive configuration options.
  - Managing application lifecycles through Helm's versioning and rollback features.
- **Apply Values Customization:**
  - Values are how changes to the manifest are made using a `values.yaml` file.
- **Managing Application Lifecycles:**
  - Incrementally roll out updates with reusable configurations.

Which is better? Kustomize or Helm?

Kustomize or Helm?

- When to Use **Helm**:
  - You need a package manager with pre-built charts for quick application setup.
  - The application requires extensive parameterization and dynamic templating.
  - You prefer managing application releases with version control and rollbacks.
- When to Use **Kustomize**:
  - You prefer a Kubernetes-native tool that works directly with YAML manifests.
  - You need simpler overlays for environment-specific configurations.
  - Avoiding templating and keeping configurations declarative is a priority.

Functions in Argo

Syncing

Syncing an Application once an update is detected

Syncing

- Syncing in Argo CD is the process of reconciling the desired state defined in a Git repository with the actual state of resources in the Kubernetes cluster.
  - **Desired State:** Kubernetes manifests stored in the Git repository.
  - **Live State:** Resources currently running in the cluster.
  - Syncing ensures the live state matches the desired state.

Types of Syncing

- **Manual Sync:**
  - Triggered by the user through the UI or CLI.
  - Allows control over when synchronization occurs.
- **Automatic Sync:**
  - Configured in the application’s sync policy.
  - Automatically applies changes when updates are detected in the repository.

SYNCING IS MATCHING THE MANIFEST IN GIT WITH THE STATE IN KUBERNETES

Syncing an Application on the CLI

If you wish to sync an application, from the CLI you can do simply with `sync`

```
$ argocd app sync <APP_NAME>
```

Rollbacks

How do we rollback to a previous rollout?

Rollbacks

Rollbacks in Argo CD can be done using several strategies, depending on whether you want to:

- Revert changes in **Git**
- Leverage **Kubernetes deployment rollbacks**
- **Manual Synchronization** to Previous Git Commit

Rollback via `git revert`

- Identify the problematic commit in the Git repository.
- Use `git revert` to create a new commit that undoes the changes

```
$ git revert <commit-hash>
```

- Push the reverted commit to the repository

```
$ git push origin <branch>
```

- Argo CD will detect the change and synchronize the application to the reverted state.

Advantages of performing `git revert`

- Ensures Git remains the source of truth.
- Provides an auditable history of changes.
- Re-applies the desired state across all clusters.

Tradeoffs of performing `git revert`

- Requires knowledge or `revert` for everyone involved
- Requires knowledge of differences between `reset` and `revert`
- May require multiple `reverts` to get to the state required
- May be difficult if git hygiene is up to quality standards
- Slower compared to on-the-fly Kubernetes rollbacks

Rollback via Kubernetes Rollout

- Use the Argo CD UI or `kubectl` to view the deployment history
- Identify the previous replica set or deployment revision.
- Roll back using the Kubernetes CLI

```
$ kubectl rollout undo deployment/<deployment-name> --to-revision=<revision>
```

Advantages of Kubernetes Rollouts

- Fast and effective for deployment-related issues.
- Useful for quick fixes without altering Git.

Tradeoffs of Kubernetes Rollouts

- Changes made directly to the cluster can lead to drift unless corrected in Git.
- Use this as a temporary measure; update Git later to ensure consistency.

Manual Synchronization to a Previous Git Commit

- In the Argo CD UI, navigate to the **Application Details**.
- Under **Sync**, select a specific Git commit or tag to deploy.
- Synchronize the application to roll back to the desired state.

Advantages of Manual Synchronization

- Allows precise control over which state to revert to.
- Useful for debugging and incremental rollbacks.

Tradeoffs of Manual Synchronization

- The rollback is effective only until the Git repository is updated.
- Future synchronizations may overwrite changes unless Git is reverted.

Pruning

Removing no longer required resources

Pruning

- Pruning in Argo CD refers to the automatic deletion of Kubernetes resources that are no longer defined in the source repository.
- Ensures that the live cluster state matches the desired state defined in the Git repository.
- During synchronization, Argo CD detects resources in the cluster not defined in the source repository and deletes them.

```
$ argocd app sync my-app --prune
```

Self Healing

Kind of what we humans do...to a point

Self-Healing

- Self-healing is a feature in Argo CD that automatically detects and corrects drift between the desired state (as defined in Git) and the live state (running in the Kubernetes cluster).
- **If a resource is modified or deleted outside of Argo CD** (e.g., manual kubectl changes), self-healing automatically re-applies the correct configuration from Git.
- Only detects and corrects resources managed by the specific Argo CD application.
- Requires a correctly configured Git repository to avoid propagating misconfigurations.
- May interfere with intentional manual changes unless those changes are reflected in Git.

Lab: Performing GitOps

- Let’s commit some changes to our forks
- Let’s then submit a pull request where an approval can take place
- Let’s observe the changes to our CI/CD environment

Strategies for Releasing to Productions

Releasing into Production in Argo CD with GitOps

- Releasing changes into production in an Argo CD and GitOps workflow can be done using various strategies
- Each strategy depends on team preferences, governance policies, and CI/CD pipelines.
- Here are the most common approaches:
  - Promotion by Manifest Updates
  - Branch based promotion
  - Tag based promotion
  - Directory based environment separation

Promotion by Manifest Updates

- The team lead or an authorized user manually updates the production environment manifests (e.g., Kubernetes YAML or Kustomize overlay) in the Git repository to reflect the desired state.
- Simple and manual. Greater control, less cognitive load
- Prone to human error if proper reviews are not enforced.
- Manual updates can slow down release velocity.

Branch Based Promotion

- Use separate Git branches for staging and production (e.g., staging and production).
- Promote changes by merging commits from staging to production.

```
$ git checkout production
$ git merge staging
$ git push origin production
```

- Clear Separation of Environments
- Requires maintaining separate branches, which can introduce merge conflicts.

Tag Based Promotion

- Use Git tags to mark stable releases for production.
- Argo CD is configured to deploy resources based on specific tags (e.g., `v1.0.0`, `prod-release`).

```
$ git tag -a v1.0.0 -m "Production release v1.0.0"
$ git push origin v1.0.0
```

- Simple and immutable tagging provides traceability.
- Avoids managing multiple branches
- Higher learning curve for those not familiar with tags

Directory Based Promotion

Use separate directories in the Git repository for each environment (e.g., staging and production).

1. Validate changes in the staging directory (e.g., overlays/staging).
2. Copy or update changes into the production directory (e.g., overlays/production).
3. Commit and push changes to Git.

Directory Based Promotion

```
overlays/
  staging/
    kustomization.yaml
    patch-deployment.yaml
  production/
    kustomization.yaml
    patch-deployment.yaml
```

- Clear separation between environments.
- Works well with tools like Kustomize.
- Requires careful synchronization of changes between directories.
- Can lead to duplication if not managed properly.

Automated Promotion using CI/CD

- Use a CI/CD pipeline to automate the promotion of changes from staging to production.
  1. Validate changes in staging using automated tests.
  2. The pipeline automatically updates the production environment manifest or merges changes to the production branch/tag upon approval.
  3. Argo CD syncs the production cluster based on the updated repository state.
- Fast and consistent.
- Reduces human error with automated checks and approvals.
- Requires pipeline setup and maintenance.
- Relies heavily on automation, which might obscure manual control.

Lab: Promote to Production

- After we have seen some changes, let us promote changes to production using Kustomize

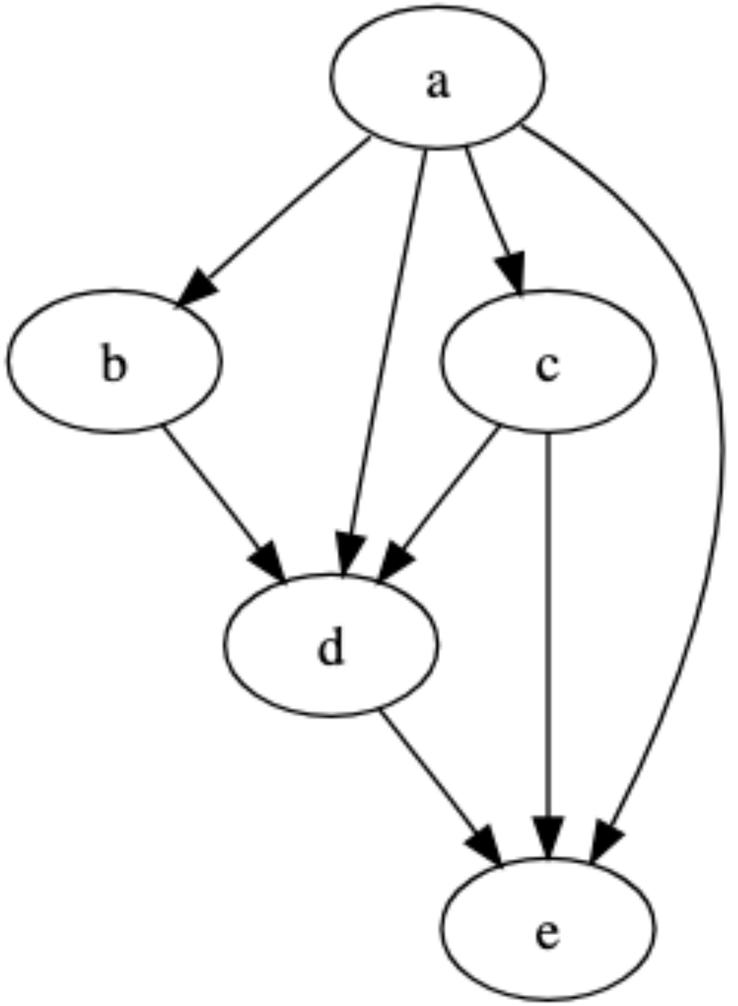
Argo CD Workflows

Argo CD Workflows

- Argo Workflows is an open-source container-native workflow engine for orchestrating parallel jobs on Kubernetes.
- It allows you to define workflows as YAML manifests, where each step runs in a Kubernetes pod.
- Automate complex job orchestration (e.g., CI/CD pipelines, data processing, and machine learning tasks).
- Enable declarative, containerized, and reproducible workflows.

Key Components of Argo Workflows

- **DAG (Directed Acyclic Graph):** Define workflows as a sequence of steps or a dependency graph.
- **Scalability:** Runs tasks in parallel on Kubernetes, leveraging container scalability.
- **Reusability:** Use templates to define reusable workflow steps.
- **Native Kubernetes Integration:** Uses Kubernetes CRDs (Custom Resource Definitions) to manage workflows.
- **Rich Artifact Management:** Support for passing artifacts between steps (e.g., files, Docker images).
- **Failure Handling:** Built-in retries, timeouts, and conditional execution.
- **User Interface:** Provides a web UI for visualizing and managing workflows.



A Directed Acyclic Graph

Demo: Argo CD Workflows

- Let’s view what ArgoCD Workflows can provide

Argo CD Events

Argo CD Events

- Argo Events is an event-driven workflow automation tool within the Argo ecosystem.
- It enables triggering Kubernetes workflows (e.g., Argo Workflows or other actions) based on external or internal events.
- Automate workflows, deployments, and application updates in response to specific events.
- Integrates with event sources like Git, Webhooks, S3, Kafka, and more

Key Components

- **Event Source:**
  - A GitHub push event is detected in the staging branch.
- **Sensor:**
  - Checks if the branch is eligible for deployment.
- **Trigger:**
  - Starts an Argo Workflow to build and deploy the application to the staging environment.

Real-world Applications of Argo Events

- **CI/CD Automation:** Trigger builds, tests, or deployments based on code changes.
  - Docker image pushed to registry
- **Cloud Integration:** Automate workflows when cloud events occur.
  - Example: Process files uploaded to S3 or GCS buckets.
- **Incident Response:** Respond to alerts or failures in real time.
  - Example: Trigger workflows based on metrics from Prometheus.
- **Notifications and Alerts:**
  - Send Slack messages, emails, or webhooks in response to events.

Demo: Argo CD Events

- Let’s view what ArgoCD Events can provide

Argo CD Rollouts

Argo CD Workflows

- Argo Rollouts is a Kubernetes controller and a progressive delivery solution that manages advanced deployment strategies.
- It extends Kubernetes Deployments and supports strategies like canary releases, blue-green deployments, and progressive rollouts.
- Enable safe and controlled updates to applications in Kubernetes clusters.
- Minimize risks during deployments by gradually rolling out changes and validating them.

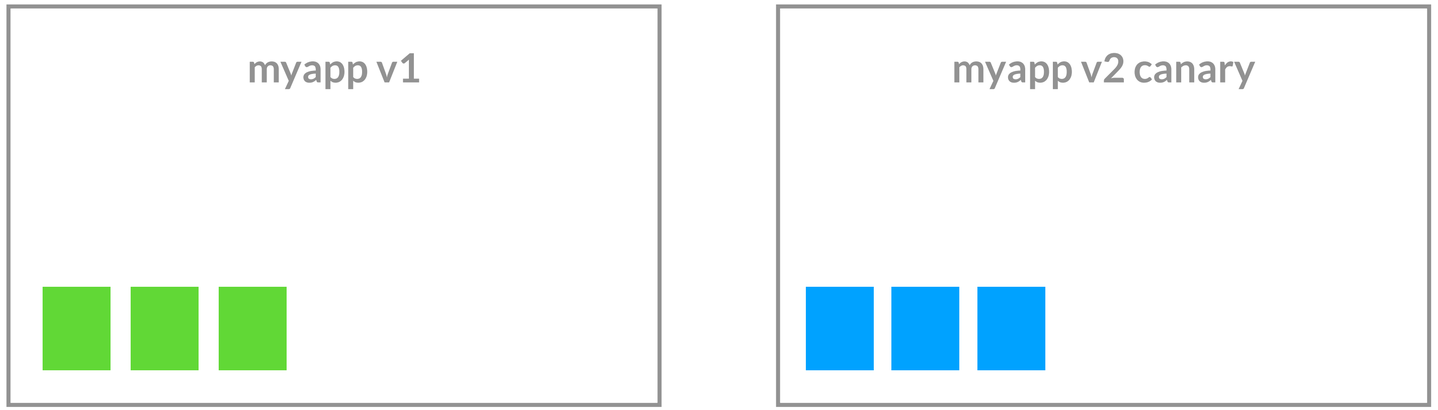
Key Components

- **Advanced Deployment Strategies:**
    - **Canary:** Gradually shift traffic to the new version while monitoring health.
    - **Blue-Green:** Deploy a new version alongside the old version and switch traffic once validated.
- **Traffic Management**
- **Rollback Support**
- **Observability**

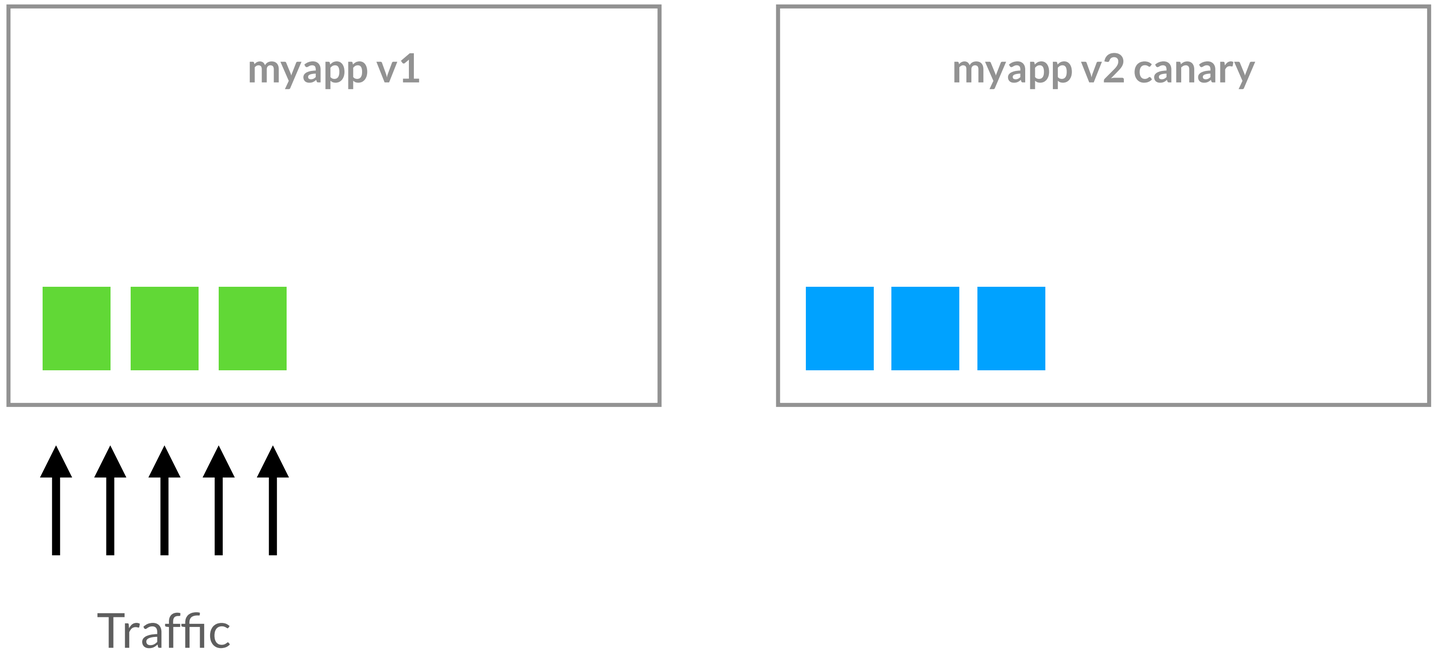
Blue Green Rollouts

Prepare and Release

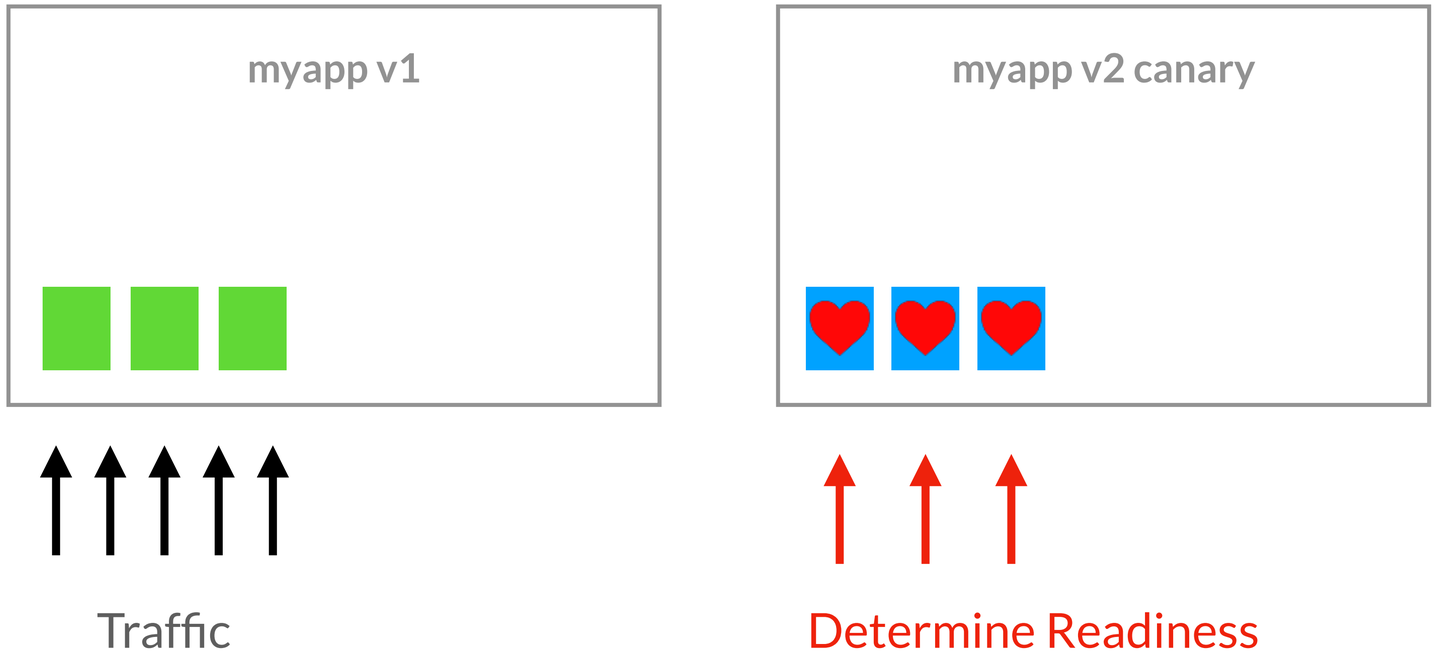
Blue Green Deployments



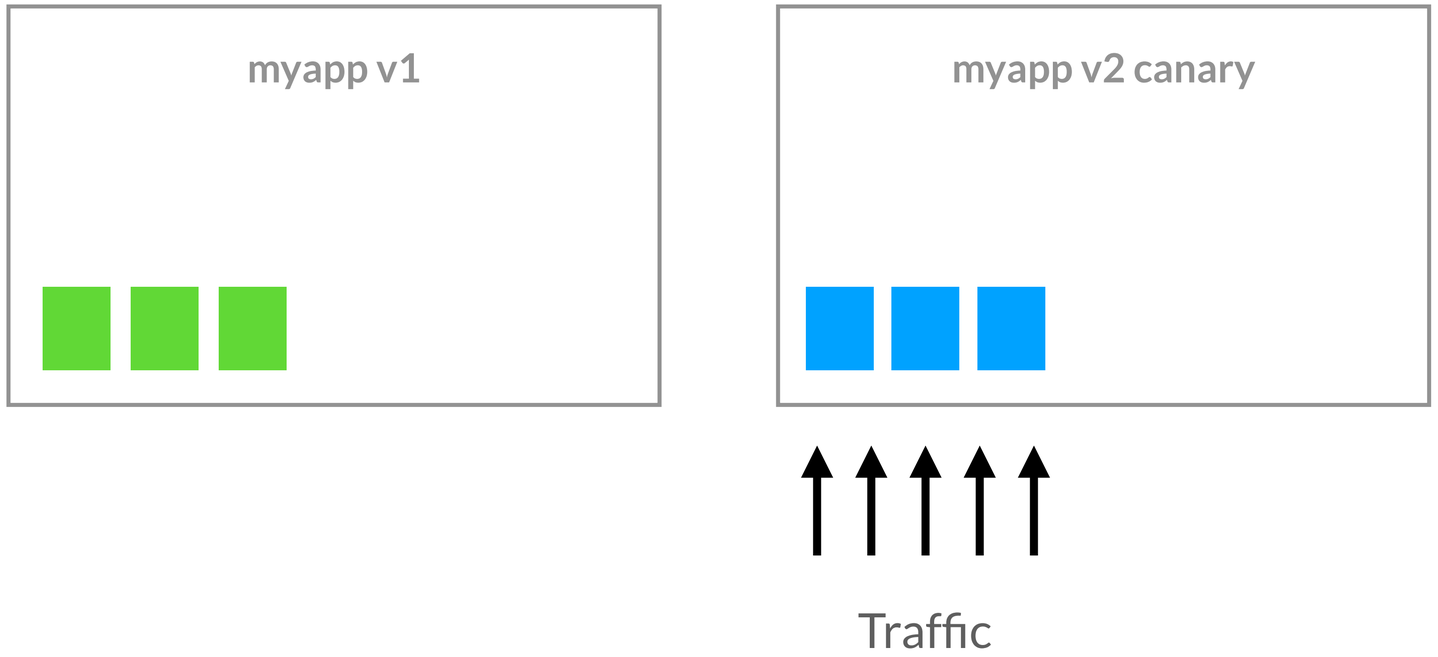
Blue Green Deployments



Blue Green Deployments



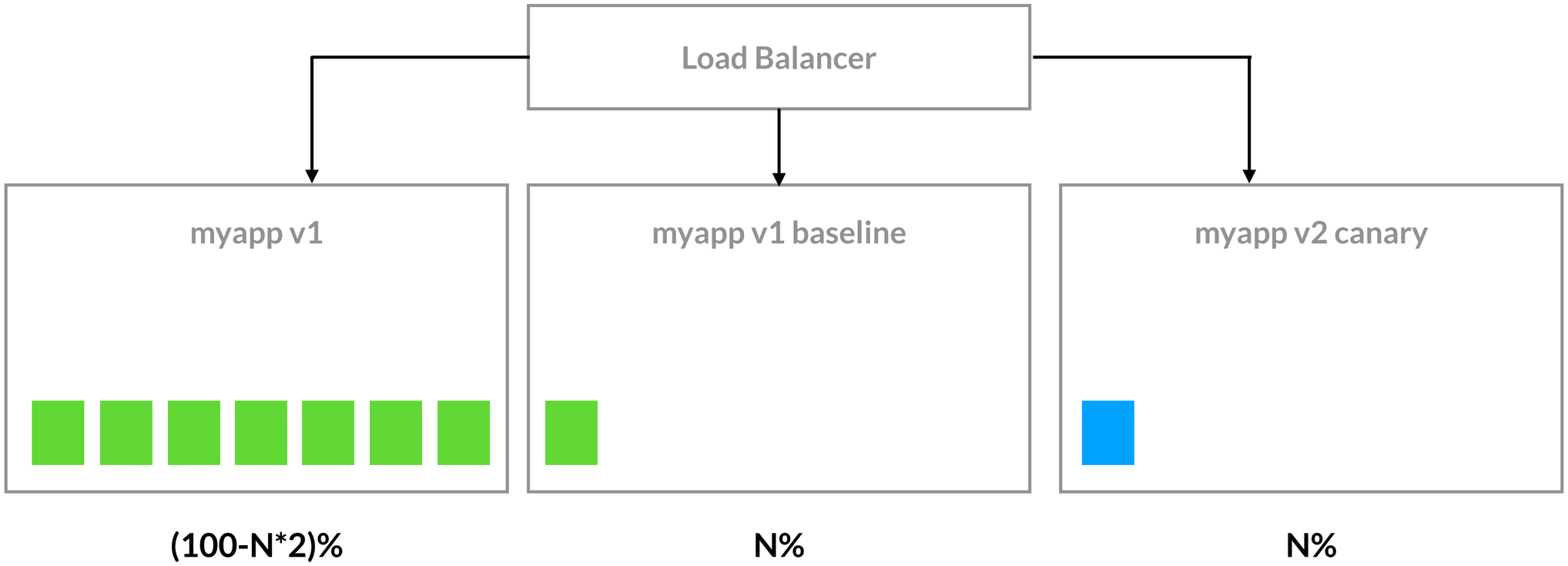
Blue Green Deployments



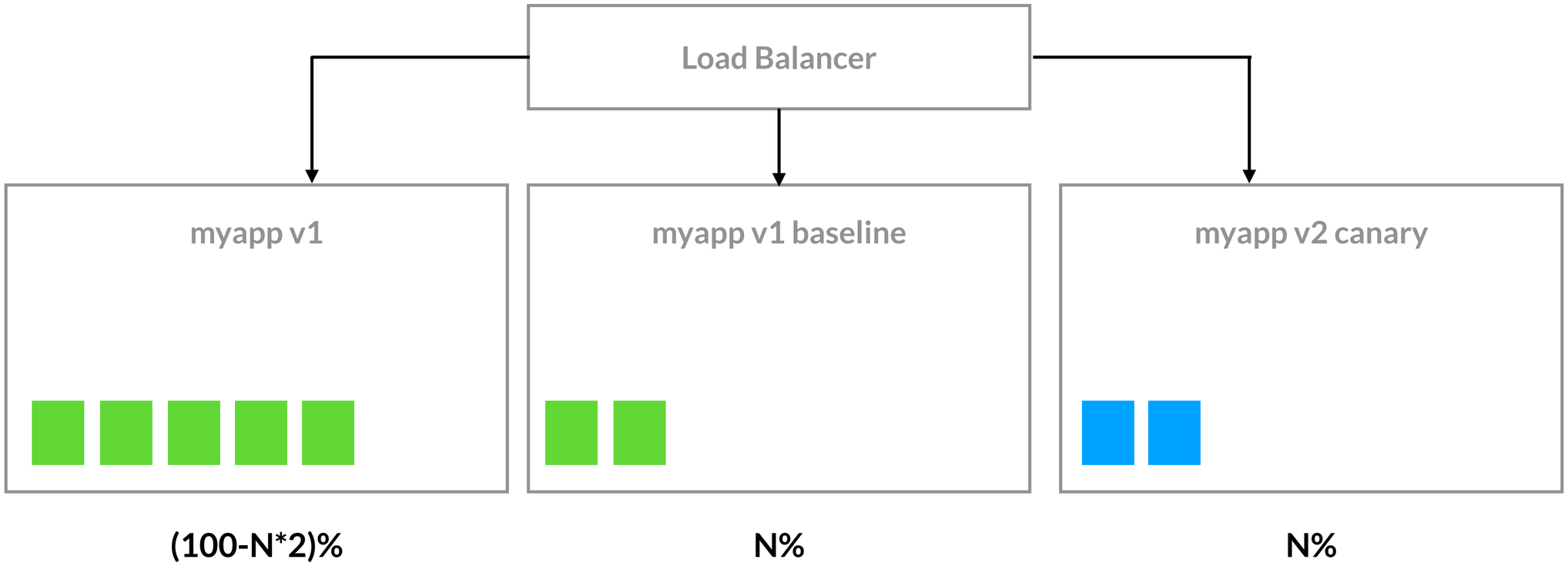
Canary Rollouts

Rolling out with Measurement

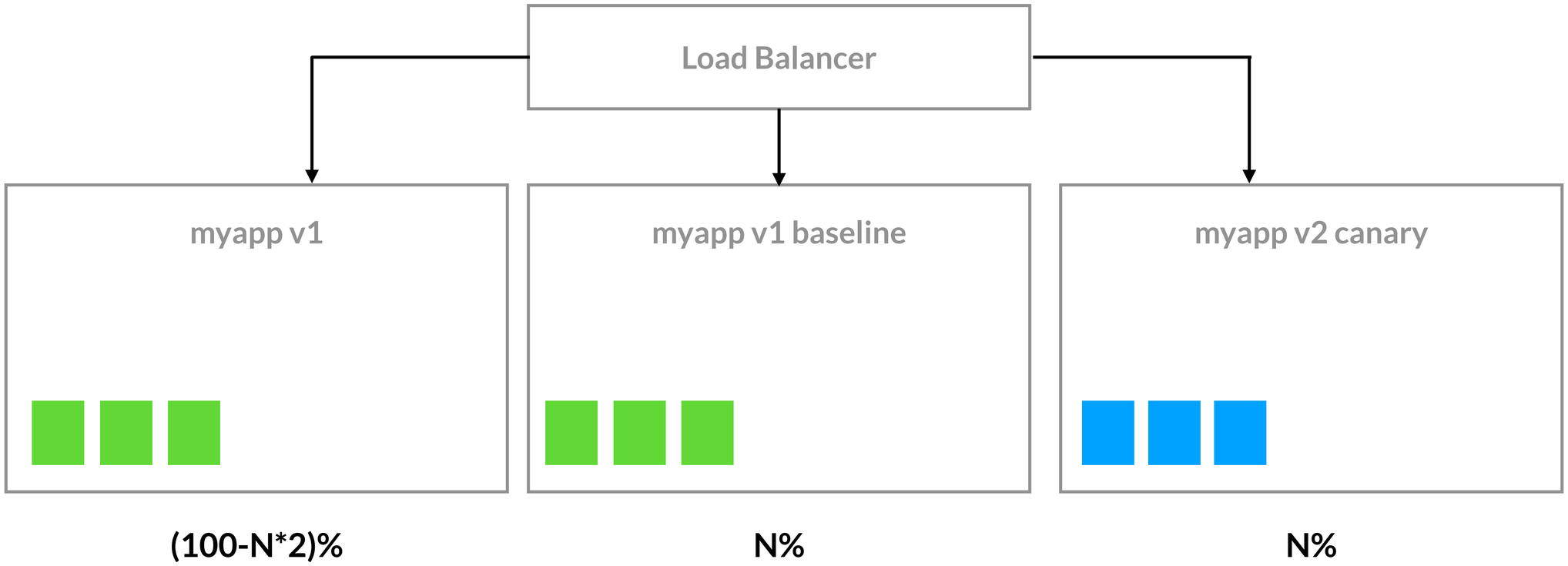
Canary Deployments



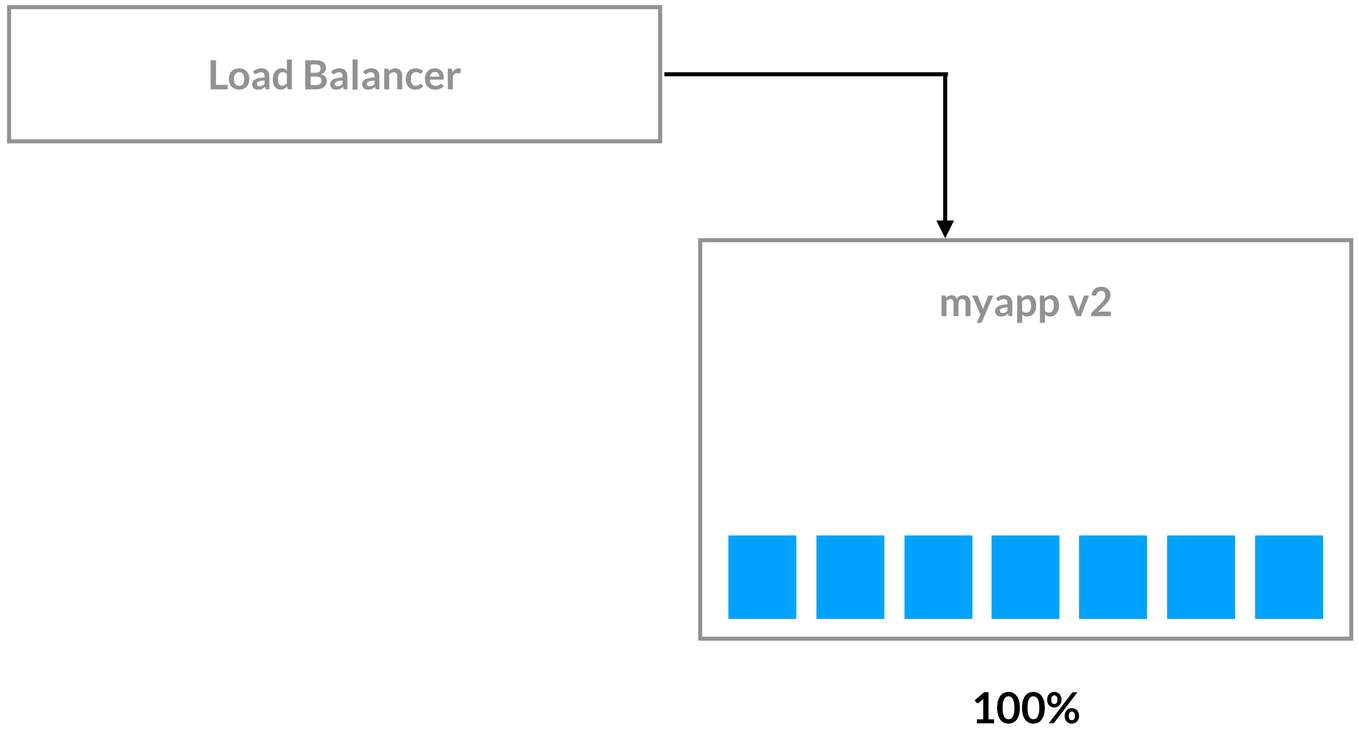
Canary Deployments



Canary Deployments



Canary Deployments



Demo: Argo CD Events

- Let’s view what ArgoCD Rollouts Can Provide

Thank You

- Email: dhinojosa@evolutionnext.com
- Github: https://www.github.com/dhinojosa
- Mastodon: https://mastodon.social/@dhinojosa
- Linked In: http://www.linkedin.com/in/dhevolutionnext