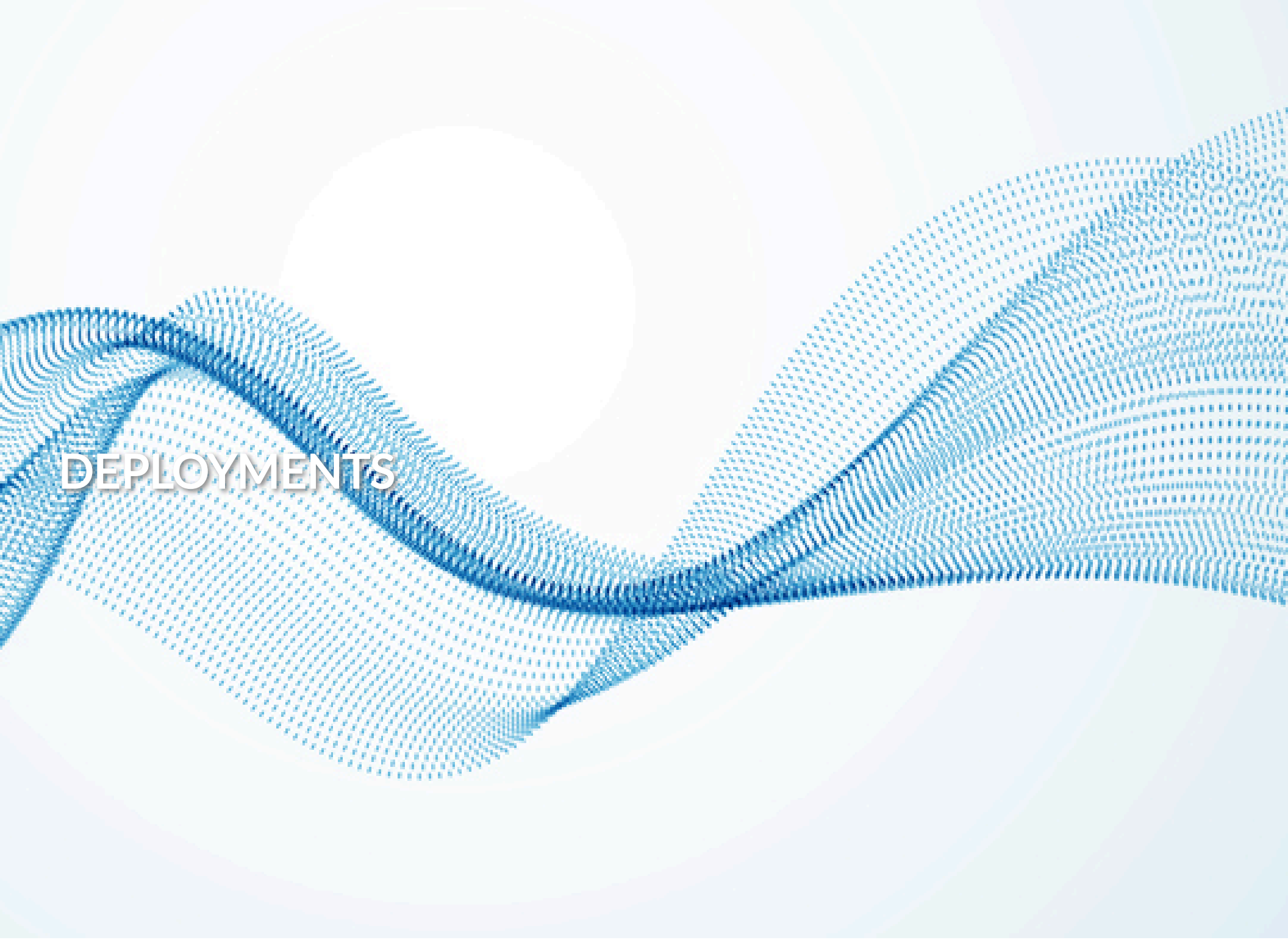

THE FINE ART OF CANARY DEPLOYMENTS
Daniel Hinojosa


DEPLOYMENTS


DEPLOYMENT TYPES

# TYPE OF DEPLOYMENTS

### Blue-green deployment

You serve the current app on one half of your environment (Blue) and deploy your new application to the other (Green) without affecting the blue environment.

### Recreate

Terminate the old version and release the new one

### Rolling Update/Ramped Update

Updates to your application are done gradually, “ramping up” new pods while slowly phasing out the old ones. It ensures that only a portion of the application instances are updated at a time, reducing the risk of downtime or service disruption.

### Canary deployment

You cut over just a small subset of servers or nodes first, before performing a full rollout. *This will be our focus*


KUBERNETES DEPLOYMENT TYPES

# KUBERNETES ROLLOUT STRATEGIES

- Kubernetes Deployments uses ReplicaSets as it’s history base
- Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate
- If the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.
- Kubernetes allows for rollbacks and scaling with Deployments
- Kubernetes also allows for cleanup, pausing, and reading the status of Deployments

# DEMO: DEPLOYMENT TYPES IN KUBERNETES

In the *canary-app* project, let's navigate to the *deployment-types*. View the manifests and let's run the manifests


CANARY DEPLOYMENTS


STRANGLER FIG

# STRANGLER FIG

- Strangler fig is the common name for a number of tropical and subtropical plant species in the genus Ficus
- These all share a common "strangling" growth habit that is found in many tropical forest species, particularly of the genus Ficus. This growth habit is an adaptation for growing in dark forests where the competition for light is intense.
- Strangler figs suck up the nutrients from its victims, causing them to die eventually.

# STRANGLER FIG PATTERN

- Incrementally replace specific pieces of functionality with new applications and services.
- Create a façade that intercepts requests going to the backend legacy system.
- The façade routes these requests either to the legacy application or the new services.
- Existing features can be migrated to the new system gradually, and consumers can continue using the same interface, unaware that any migration has taken place.


CANARY

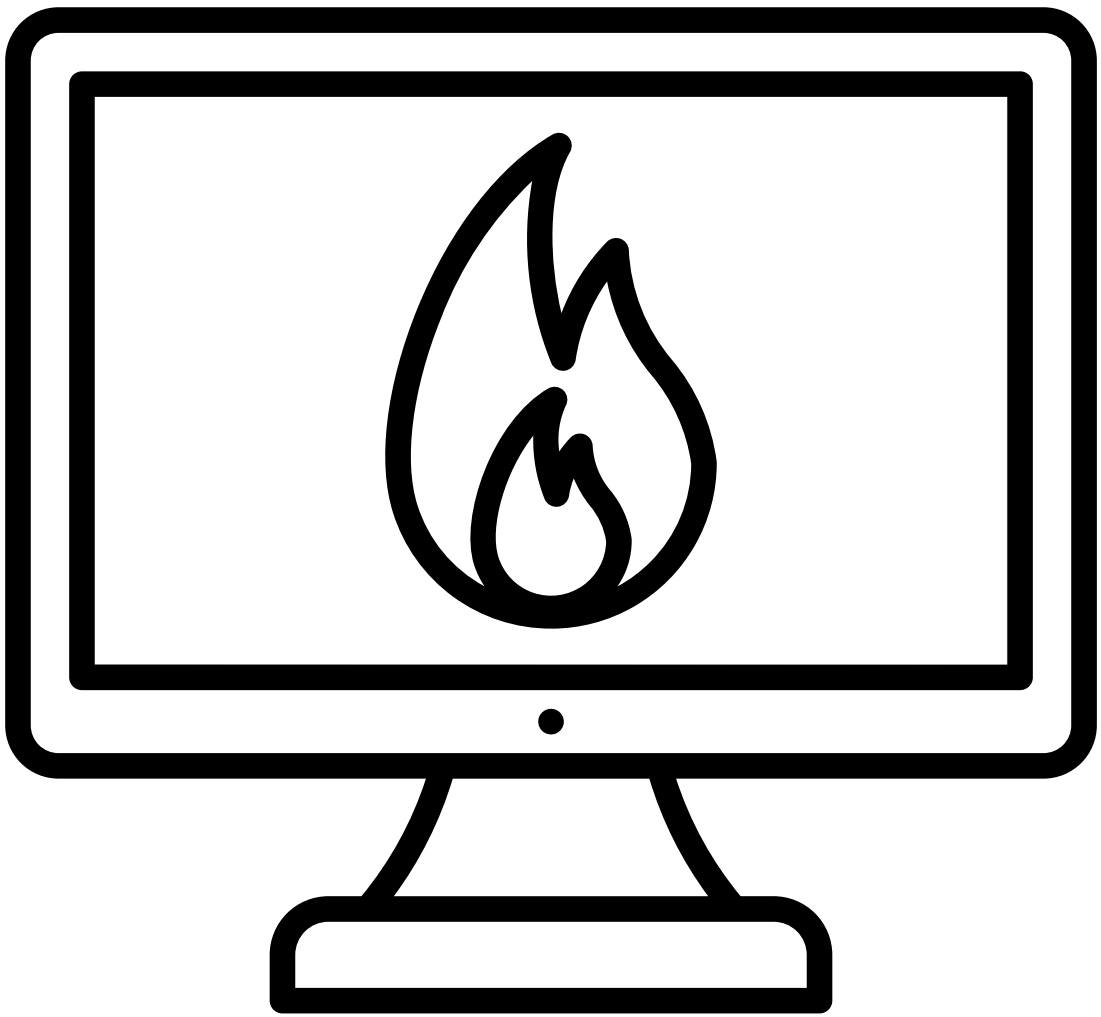
# CANARY IN A COALMINE

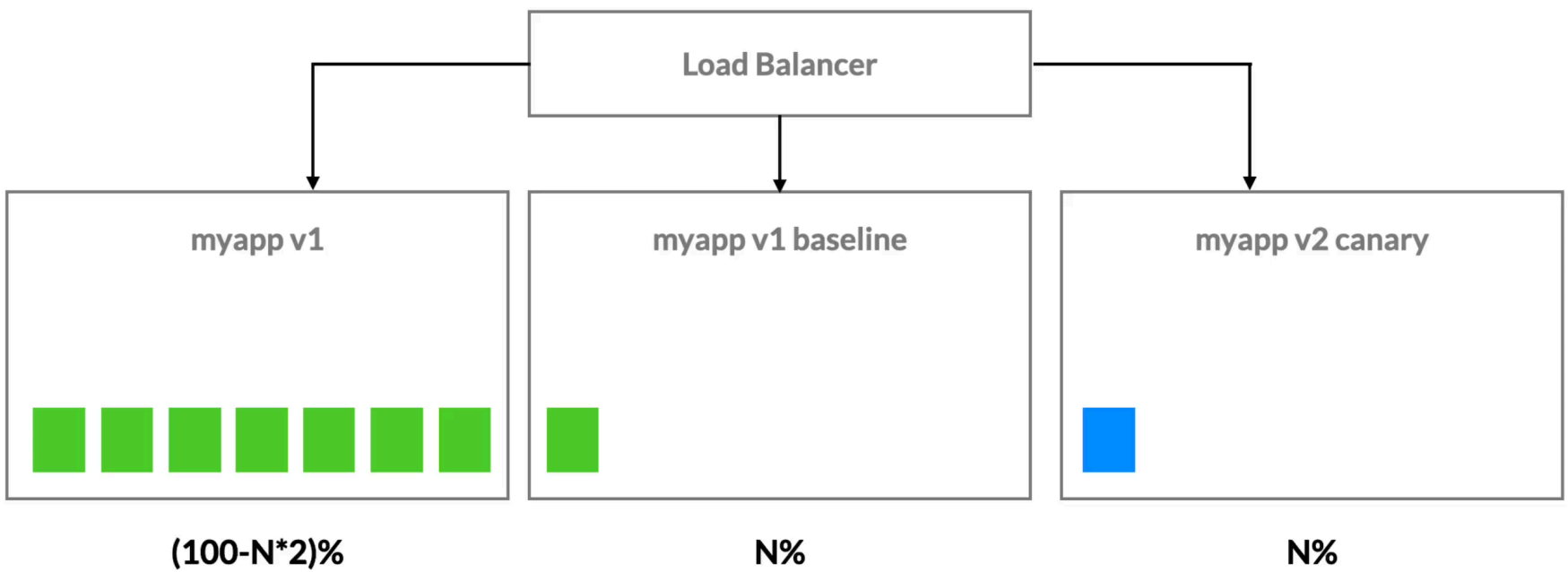
- *Def:* An early indicator of potential danger or failure.
- *Eg:* "The native brook trout are very much the canary in the coal mine for the health of a stream"
- Canaries have smaller lung capacity.
- If the canaries are dead, then $CO_2$ levels were too high
- The practice was stopped in 1986 and replaced with sensors


THE POLICE
ZENYATTÀ MONDATTA

# WHAT IS CANARY DEPLOYMENT?

- Canary is a deployment process in which a change is partially rolled out ***in production***, then evaluated against the current deployment (baseline)
- Ensure that the new deployment is operating at least as well as the old.
- This evaluation is done using key metrics that are chosen when the canary is configured.
- Microservices would need to expose some of their metrics
- Canaries are usually run against deployments containing changes to code, but they can also be used for operational changes, including changes to configuration.
- No "big bang" release




Load Balancer
myapp v1
myapp v1 baseline
myapp v2 canary
(100-N*2)%
N%
N%

# WHAT IS A BASELINE?

- In a canary deployment or canary release, the ***"baseline"*** refers to the existing, stable version of a software application or service that is already running in production.
- Its purpose is to serve as a reference point for comparison when deploying a new version (the "canary").

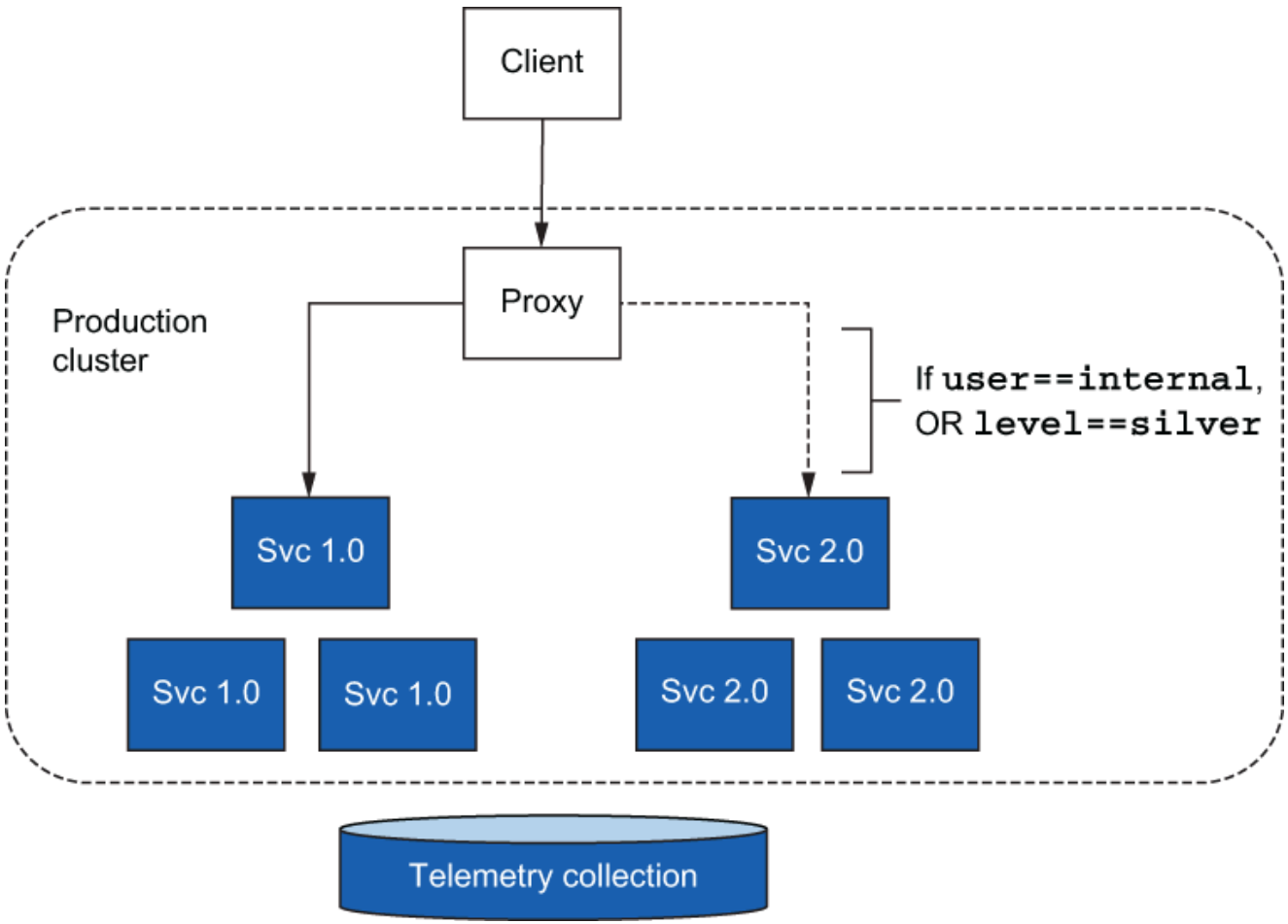
# WHAT IS THE PURPOSE OF A BASELINE?

- Isolate a portion of production to compare with the canary
- Instead of duplicating the entire production environment, only a portion is dedicated to comparison, which:
  - Minimizes costs
  - Minimizes resource usage
  - Creates a representative sample
  - Ability to perform A/B Testing

# THE CASE FOR HIDDEN DEPLOYMENTS AND SHADOWING

- Strategy used in software development to test a new version of an application in a production environment without affecting the end users.
- The key idea behind shadow deployments is to send live production traffic to the new version of the application (the “shadow” version), while still routing user-facing traffic to the current stable version.
- However, the responses from the shadow version are not returned to users; they are only used for testing and monitoring purposes.

# DIAGRAM OF SHADOW DEPLOYMENTS




Source: Istio in Action Rinor Maloku, Christian E. Posta Published by Manning Publications

# A/B TESTING

- When we have a new release and an old release we can perform A/B Testing.
- Where the new release, or canary is the "experiment" group and the baseline is the "control" group
- We can then test to see if a new feature is equal, less, or more effective that what it currently is.
- For the results of an A/B test to be reliable, they must be statistically significant.
    - This means the differences observed between A and B are unlikely to have occurred by chance.
    - Statistical tools and techniques like p-values, z-values, confidence variables and other statistical functions that would be necessary here to determine if the results are significant enough

# A/B TESTING EXAMPLE

We hypothesize that changing the “Buy Now” button color from blue to green will lead to higher conversions


HOW CAN WE PERFORM CANARY DEPLOYMENTS?


ENVOY

# WHAT IS ENVOY?

- Envoy is a level 7 proxy as defined by OSI Networking Model (HTTP, FTP, IRC, SSH, DNS)
- Developed by Lyft that offers services before sending to actual services
- It is an ***Ambassador*** Architectural pattern
- It is deployed along with a client to provide numerous services

# WHAT KIND OF SERVICES?

- **Circuit Breaking** - Provides the ability to back off from calling services that are suffering latency or is down
- **Observability** - Provides observability data to be accumulated by software that does something with that data (e.g. Prometheus, Datadog). Provides tracing that also can be accumulated by an aggregator (e.g. Jaeger, Datadog)
- **Security** - Envoy supports both TLS termination in listeners, including TLS origination when making connections to upstream clusters
- **Load Balancing** - Routes traffic with varying strategies like Round Robin, Least Request, and Random. Include Traffic Splitting which allows us to perform canary.

# CANARY WITH ENVOY USING WEIGHTS

```yaml
routes:
  - match:
      prefix: "/"
    route:
      weighted_clusters:
        clusters:
          - name: service_a_cluster
            weight: 80
          - name: service_b_cluster
            weight: 20
```

See Documentation to show more about weighted clusters

# DEMO: ENVOY PROXY

In the *canary-app* project, let's navigate to the *envoy*. We will run the *docker-compose.yaml* file and run against two services


KUBERNETES IN THE RAW

# COMPONENT REVIEW

### Kubernetes Deployment

- Manages and automates the lifecycle of Pods (i.e., container instances).
- Supports rolling updates, scaling, and rollbacks of applications.
- Ensures the desired number of replicas of Pods are running.

### Kubernetes ReplicaSet

- Ensures a specified number of Pod replicas are running at any given time.
- Replaces Pods if they fail or are terminated.
- A Deployment automatically creates and manages ReplicaSets.

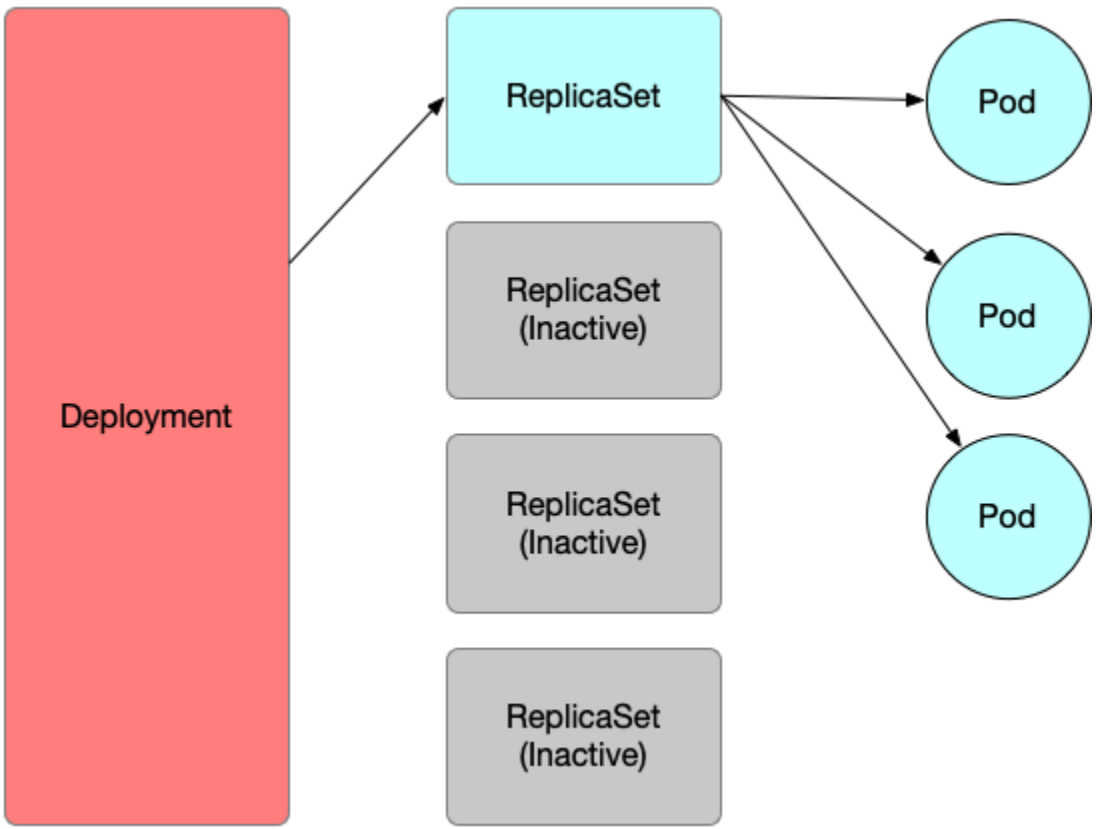
### Kubernetes Service

- Provides a stable network endpoint (e.g., DNS name) to access a group of Pods.
- Load balances traffic across the Pods that match the Service's selector.
- Types include `ClusterIP` (internal access), `NodePort` (external access), and `LoadBalancer` (cloud-based load balancing).

# `ReplicaSet`S ARE HISTORICAL RELEASES FOR A `Deployment`

- While `ReplicaSet`s can be created independently, in most modern Kubernetes usage, they are managed by `Deployment`s.
- A `Deployment` can create multiple `ReplicaSet`s over its lifetime, especially when rolling updates or rollbacks are involved.
- `ReplicaSet`s are not immediately deleted by the `Deployment`; Kubernetes retains them to support rollbacks or for history tracking.

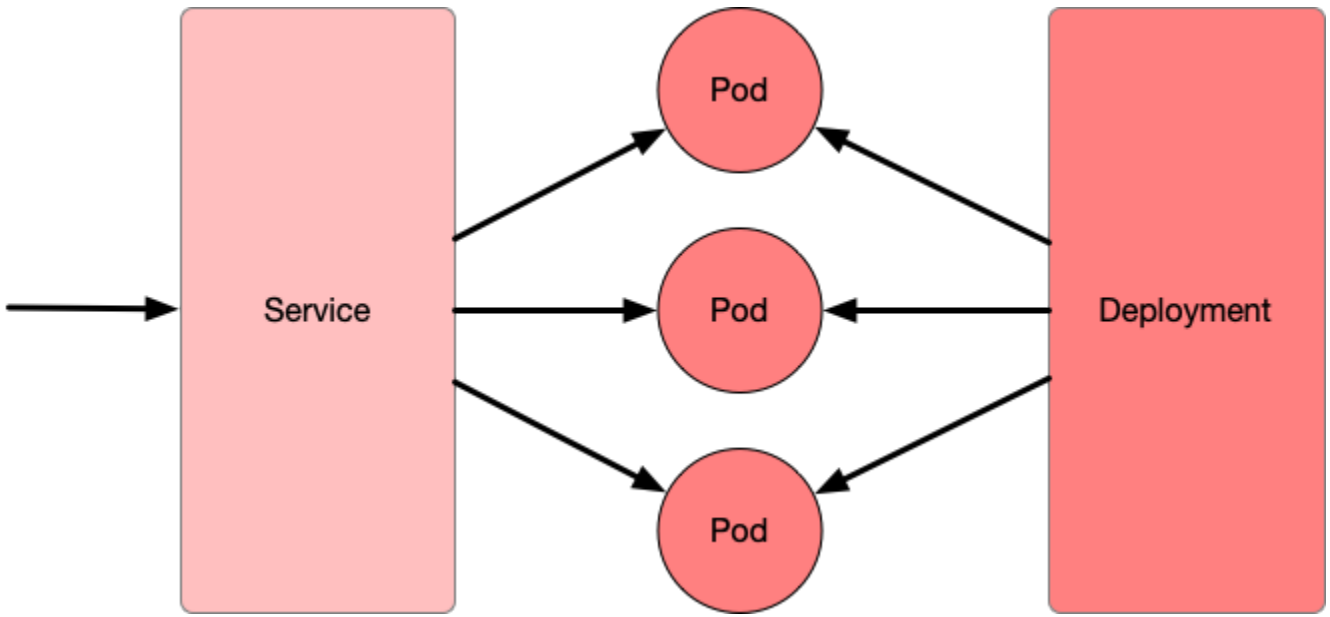
# RELATIONSHIP BETWEEN DEPLOYMENT AND REPLICA SET



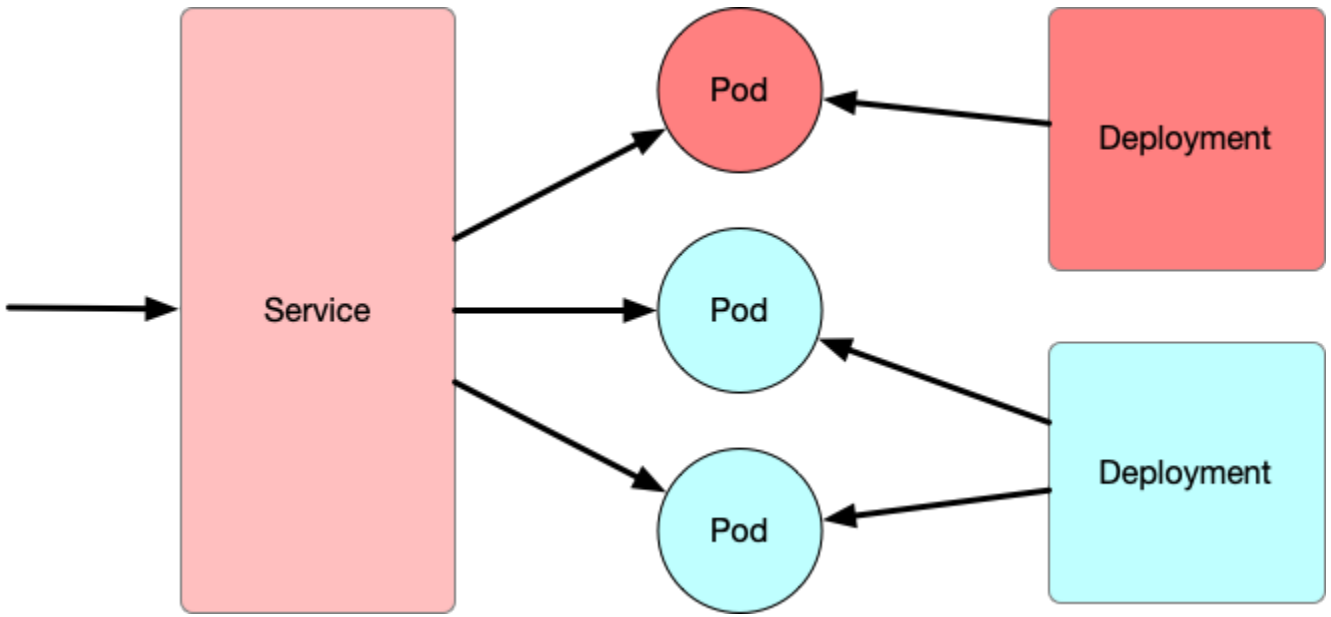
# SEPARATION OF `Service` AND `Deployment`

- A `Service` and a `Deployment` are two separate Kubernetes resources. They are not directly tied to each other and can be created independently.
- A `Service` can and *should be deployed first*.
- A `Service` is typically *fixed* and works like a service desk at your post office or favorite shipping company.
- *You don't know what is behind the service desk and how they go about their business, they just fulfill your request*

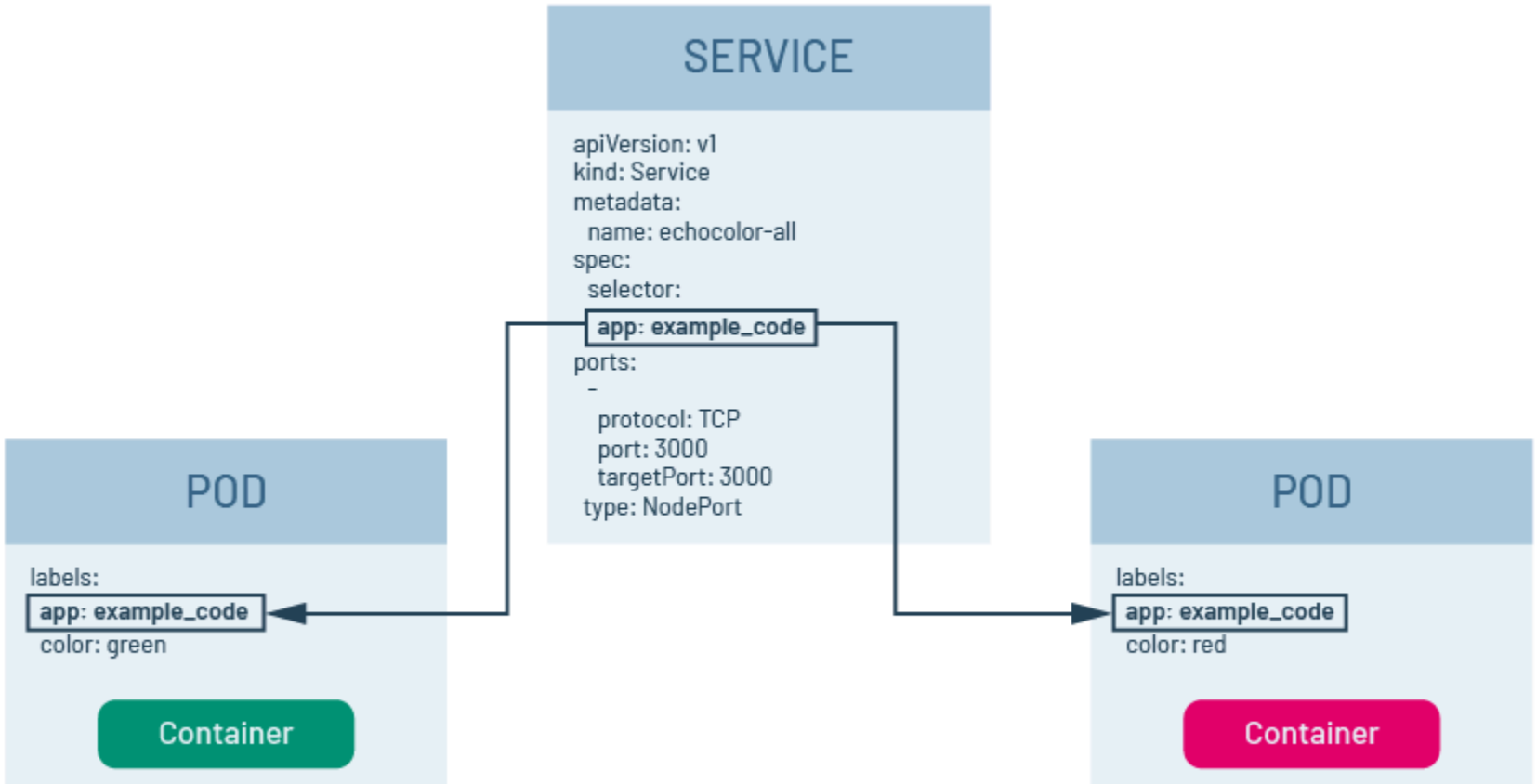
# TIGHTLY PAIRED



# DIFFERENT DEPLOYMENTS



# IT'S ALL JUST LABELS



# WHAT IS WRONG WITH DOING CANARIES THIS WAY?

- Doing it purely with counts works, but the math is on you
- Strategically plan what the counts are going to look like over the long term
- Rollbacks will need to be done to two deployments
- You will need to trim `ReplicaSet`s that are not viable

# HOW DO WE DO ROLLBACKS?

- Rolling backs will be done by rolling back a deployment
- This will roll back to a previous `ReplicaSet`
- Continuous Delivery Frameworks with a nice UI should make this easy

# DEMO: KUBERNETES RAW CANARIES

In the *canary-app* project, let's navigate to the *kubernetes-raw-canaries*. View the manifests and let's run the manifests

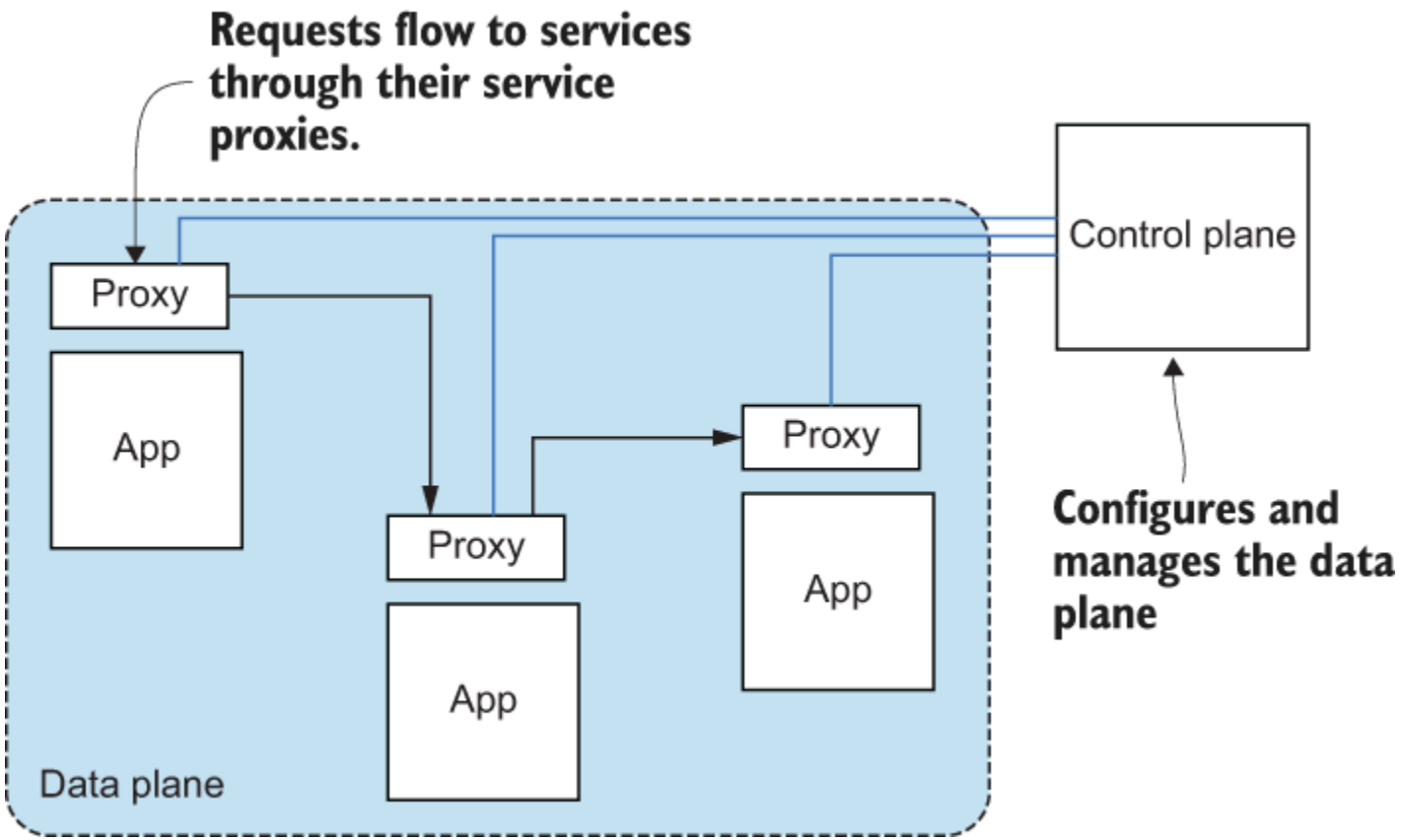

ISTIO

# WHAT IS A SERVICE MESH?

- A ***service mesh*** is a distributed application infrastructure that is responsible for handling network traffic on behalf of the application in a transparent, out-of-process manner.
- A ***data plane*** is responsible for establishing, securing, and controlling the traffic through the mesh.
- A ***control plane*** is the brains of the mesh and exposes an API for operators to manipulate network behaviors.
- Together, the data plane and the control plane provide important capabilities.

Source: Istio in Action, Rinor Maloku, Christian E. Posta

# ISTIO CONTROL PLANE AND SERVICE MESH




Source: Istio in Action, Rinor Maloku, Christian E. Posta

# WHAT IS ISTIO?

- Istio is an open source implementation of a service mesh founded by Google, IBM, and Lyft.
- It helps you add resilience and observability to your services architecture in a transparent way
- Applications don’t have to know that they’re part of the service mesh: whenever they interact with the outside world, Istio handles the networking on their behalf

Source: Istio in Action, Rinor Maloku, Christian E. Posta

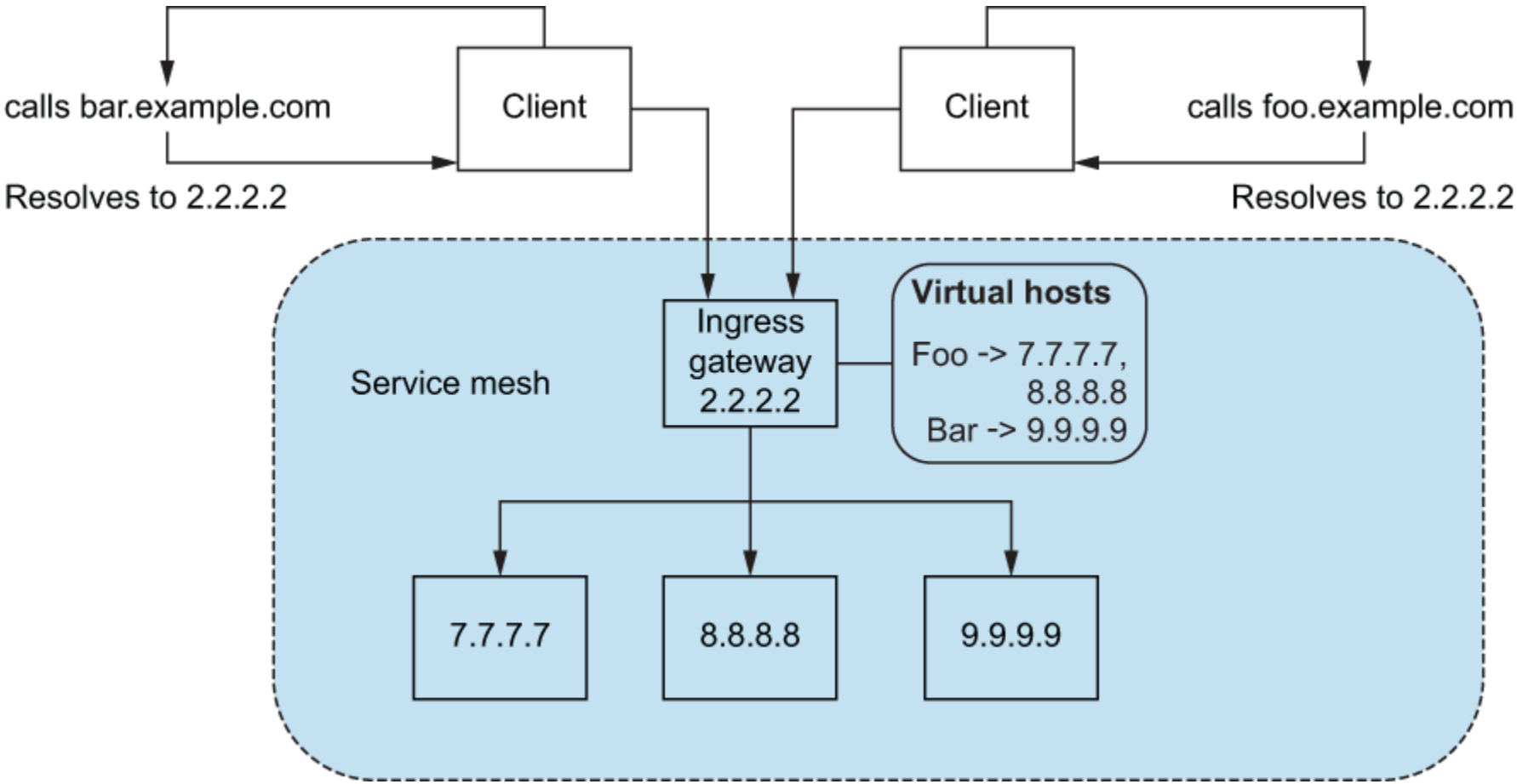
# RECIPE FOR CREATING CANARIES IN ISTIO

- Create an Istio `Gateway`
- Create an Istio `VirtualService`
- Create an optional Istio `DestinationRule`

# ISTIO INGRESS GATEWAY

- Istio has the concept of an ***ingress gateway***
- An ingress plays the role of the network ingress point and is responsible for guarding and controlling access to the cluster from traffic that originates outside the cluster.
- Internally uses an Envoy to route traffic to the necessary locations

# ISTIO INGRESS GATEWAY




Source: Istio in Action, Rinor Maloku, Christian E. Posta

# ISTIO VIRTUAL SERVICE

- Configuration affecting traffic routing.
- Defines a set of traffic routing rules to apply when a host is addressed.*
- Each routing rule defines matching criteria for traffic of a specific protocol.
- If the traffic is matched, then it is sent to a named destination service (or subset/version of it) defined in the registry.

Source: Istio Website on `VirtualService`

# ISTIO VIRTUAL SERVICE

```yaml
kind: VirtualService
metadata:
  name: helloworld
spec:
  hosts:
    - helloworld
  http:
  - route:
    - destination:
        host: helloworld
        subset: v1
      weight: 90
    - destination:
        host: helloworld
        subset: v2
      weight: 10
```

# DESTINATION RULE

- An Istio ***Destination Rule*** defines policies that apply to traffic intended for a service after routing has occurred.
- It is *advanced traffic management* to specify how requests to a service are handled.
- They include features for: Load balancing, Connection pools, and Circuit breakers
- They allow fine-grained control over traffic by grouping instances of a service into subsets (e.g., based on version or environment) and applying different policies to each subset.

# DEMO: ISTIO

In the *canary-app* project, let's navigate to the *istio*. View the manifests and let's run the manifests


INTEGRATING METRICS

# PROMETHEUS

- Open-source systems monitoring and alerting toolkit.
- It was originally built at SoundCloud
- Prometheus scrapes and stores metrics as time series data, recording information with a timestamp.
- Contains a query language (PromQL) for aggregating and analyzing metrics.
- Prometheus can generate alerts based on metric data and alerting rules.
- It integrates well with various visualization tools, notably Grafana.
- Prometheus supports service discovery and dynamic configurations through its broad range of integrations.

# PROMETHEUS ARCHITECTURE

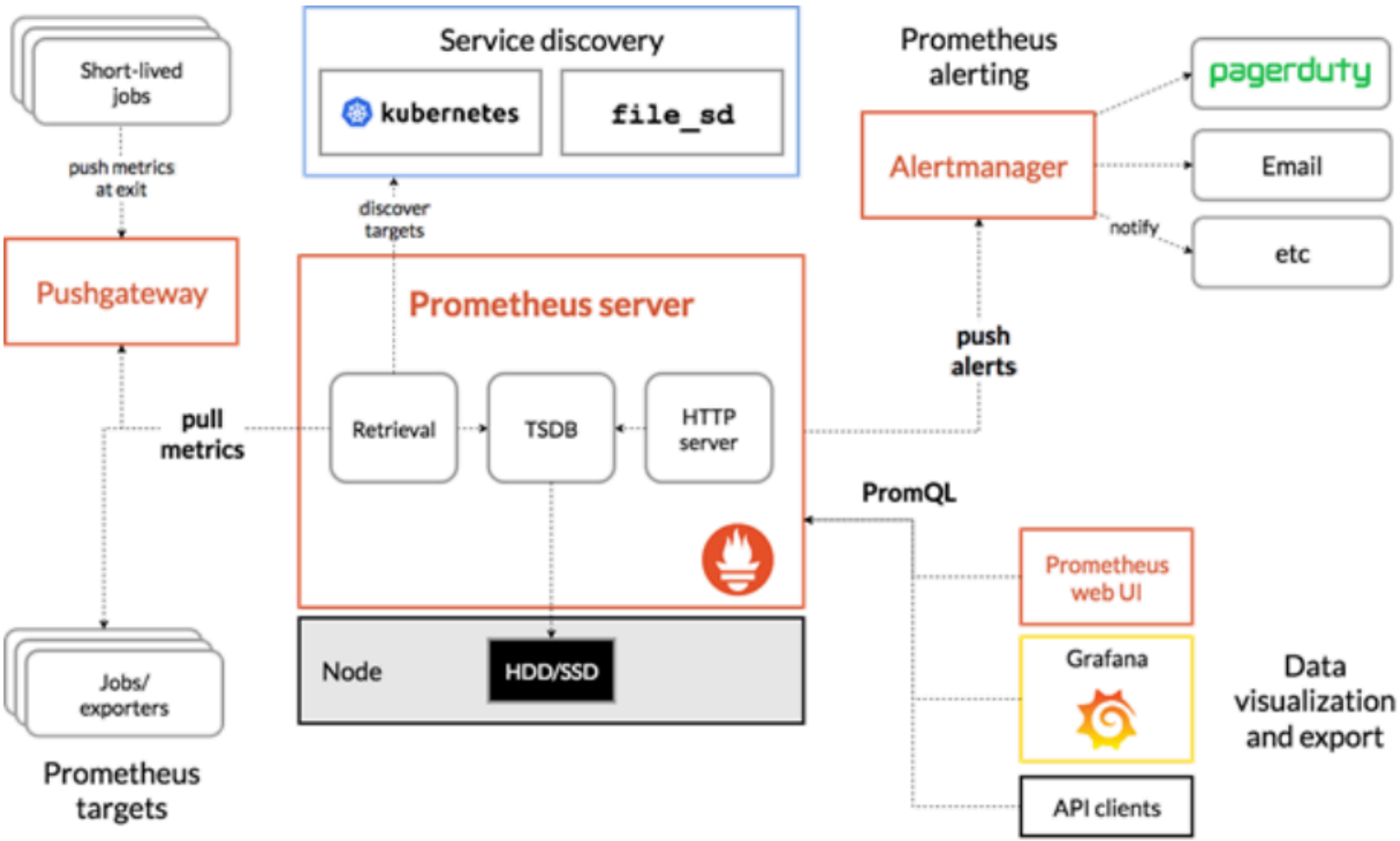
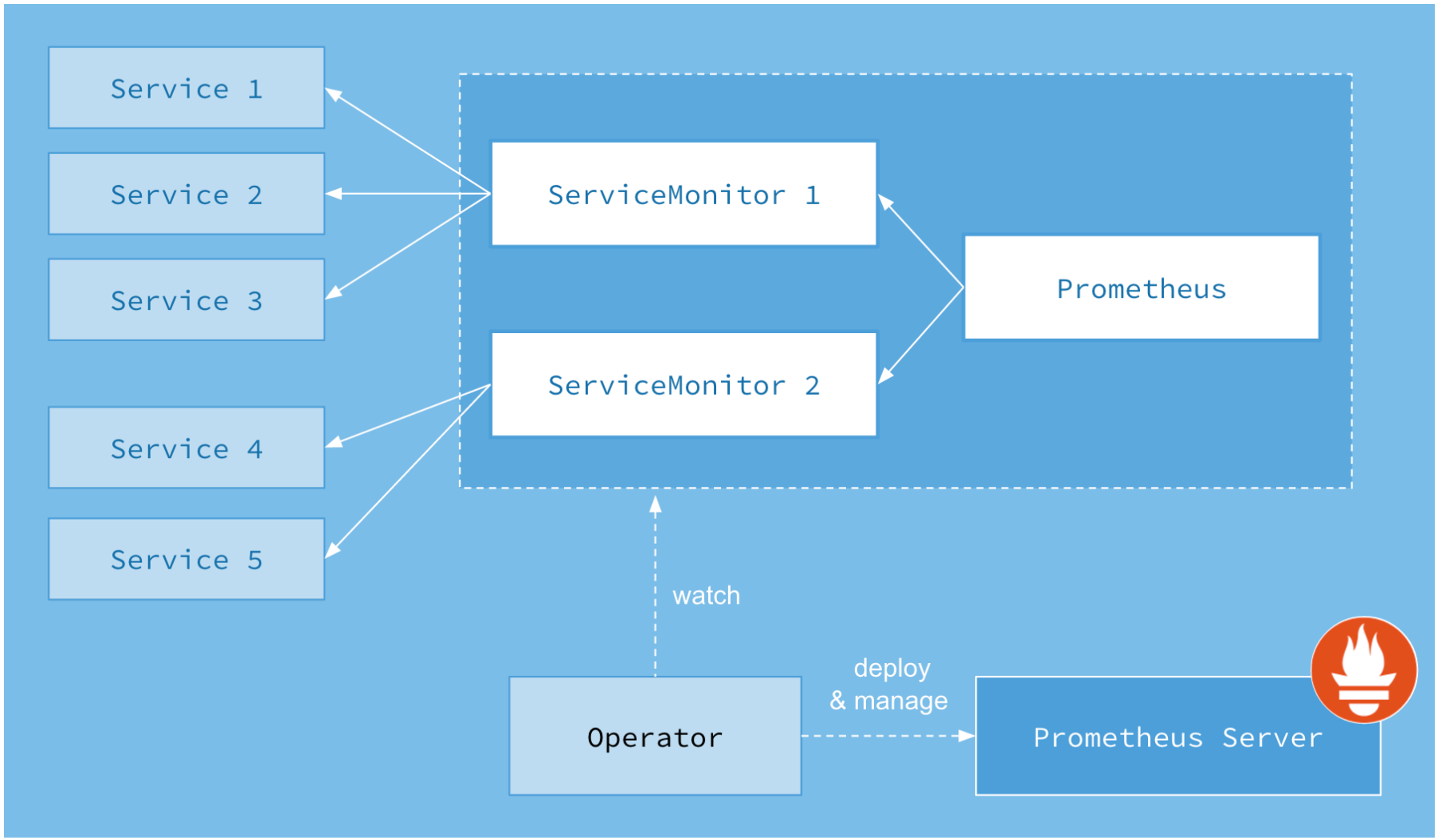



Figure 1: Prometheus architecture

# ABOUT THE PROMETHEUS OPERATOR

- The Prometheus Operator simplifies the deployment and management of Prometheus instances in Kubernetes clusters.
- It provides Kubernetes-native deployment and management mechanism for Prometheus instances.
- Automates the configuration of Prometheus, alertmanager, and related services.
- Most of all, ***It automatically installs and scrapes metrics for Kubernetes Nodes***

# PROMETHEUS OPERATOR DIAGRAM



# PROMETHEUS EXPOSITION FORMAT

- *Prometheus Exposition Format* is the standard format that Prometheus uses to scrape metrics from applications.
- It allows Prometheus to collect metrics in a human-readable plaintext format.
- The format consists of a simple text-based protocol where each metric is represented as a line of text.
- Metrics include both scalar values and event counts, tagged with corresponding labels for better querying.

# PROMETHEUS EXPOSITION FORMAT EXAMPLE

```text
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="post",code="200"} 1027 1395066363000
http_requests_total{method="post",code="400"}    3 1395066363000

# Escaping in label values:
msdos_file_access_time_seconds{path="C:\\DIR\\FILE.TXT",error="Cannot find file:\n\"FILE.TX

# Minimalistic line:
metric_without_timestamp_and_labels 12.47

# A weird metric from before the epoch:
something_weird{problem="division by zero"} +Inf -3982045

# A histogram, which has a pretty complex representation in the text format:
# HELP http_request_duration_seconds A histogram of the request duration.
```

# QUARKUS

**To expose metrics in Prometheus Exposition Format in a Quarkus application, you can use**

- **SmallRye MicroProfile Metrics** extension
- **Micrometer Metrics** extension

# ADDING MICROMETER METRICS IN MAVEN

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-micrometer-registry-prometheus</artifactId>
</dependency>
```

# ADDING MICROMETER METRICS IN GRADLE

```groovy
implementation 'io.quarkus:quarkus-micrometer-registry-prometheus'
```

# ADDING SMALL RYE MICROPROFILE IN MAVEN

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-smallrye-metrics</artifactId>
</dependency>
```

# ADDING SMALL RYE MICROPROFILE IN GRADLE

```groovy
implementation 'io.quarkus:quarkus-smallrye-metrics'
```

# USING CUSTOM METRICS IN JAVA FOR QUARKUS USING SMALLRYE

```java
import org.eclipse.microprofile.metrics.annotation.Counted;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("/example")
public class ExampleResource {

    @GET
    @Counted(name = "example_counter",
             description = "Number of times this endpoint is called")
    public String exampleEndpoint() {
        return "Hello, Quarkus!";
    }
}
```

# USING CUSTOM METRICS IN JAVA FOR QUARKUS USING MICROMETER

```java
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.inject.Inject;

@Path("/example")
public class ExampleResource {

    private final Counter exampleCounter;

    @Inject
    public ExampleResource(MeterRegistry registry) {
        this.exampleCounter = Counter.builder("example_counter")
                                     .description("Number of times this endpoint is called'
                                     .register(registry);
```

# SPRING BOOT

To expose metrics in Prometheus Exposition Format in a Spring Boot application, you can use the **Micrometer** which is included in the Spring Boot Actuator

# ADDING MICROMETER IN MAVEN

```xml
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

# ADDING MICROMETER IN GRADLE

```groovy
implementation 'io.micrometer:micrometer-registry-prometheus'
implementation 'org.springframework.boot:spring-boot-starter-actuator'
```

# ENABLING PROMETHEUS METRICS IN SPRING BOOT

properties

```
management.endpoints.web.exposure.include=prometheus
management.endpoint.prometheus.enabled=true
```

# USING CUSTOM METRICS IN JAVA FOR SPRING BOOT

```java
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import org.springframework.stereotype.Component;

@Component
public class CustomMetrics {

    private final Counter customCounter;

    public CustomMetrics(MeterRegistry meterRegistry) {
        this.customCounter = meterRegistry.counter("custom.counter");
    }

    public void incrementCounter() {
        this.customCounter.increment();
    }
```

# MICRONAUT

To expose metrics in Prometheus Exposition Format in a Micronaut application, you can use the **Micronaut Micrometer Prometheus** module.

# ADDING MICRONAUT MICROMETER PROMETHEUS IN MAVEN

```xml
<dependency>
    <groupId>io.micronaut.micrometer</groupId>
    <artifactId>micronaut-micrometer-registry-prometheus</artifactId>
</dependency>
```

# ADDING MICRONAUT MICROMETER PROMETHEUS IN GRADLE

```groovy
implementation "io.micronaut.micrometer:micronaut-micrometer-registry-prometheus"
```

# ENABLING PROMETHEUS METRICS IN MICRONAUT

To enable Prometheus metrics in your `application.yml`, configure the following:

```yaml
endpoints:
  prometheus:
    sensitive: false
micronaut:
  metrics:
    enabled: true
    export:
      prometheus:
        enabled: true
        step: PT1M
        descriptions: true
```

# USING CUSTOM METRICS IN JAVA FOR MICRONAUT

To use custom metrics in a Micronaut application, you can create a class to define and increment custom counters as follows:

```java
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import io.micronaut.context.annotation.Requires;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

import javax.inject.Inject;

@Requires(beans = MeterRegistry.class)
@Controller("/example")
public class ExampleController {

    private final Counter exampleCounter;

    @Inject
    public ExampleController(MeterRegistry meterRegistry) {
```

# WHAT METRICS ARE IMPORTANT IN CANARY DEPLOYMENTS?

# REQUEST AND RESPONSE METRICS

- **Request Rate (RPS/QPS)**: Measure the number of requests per second handled by both the canary and the baseline versions. Ensure that the canary is handling traffic as expected without introducing spikes or drops in request volume.
- **Error Rate (HTTP 4xx, 5xx)**: Track the percentage of requests returning 4xx or 5xx errors. A higher error rate in the canary service could indicate regressions or issues in the new version.
- **Success Rate**: Focus on the percentage of successful requests (200 OK or other valid response codes). A decline in success rate could mean problems with the canary version.

# LATENCY AND RESPONSE TIME

- **Latency (P50, P90, P99)**: Measure the time it takes for the canary service to respond to requests. High percentiles (P90 or P99) are especially important for identifying latency spikes that could negatively impact user experience.
- **Average Response Time**: Keep track of the average time taken by the canary to respond. If the response time for the canary version is significantly higher than the baseline, it might indicate performance bottlenecks.

# RESOURCE UTILIZATION METRICS

- **CPU Usage**: Monitor the CPU usage of the canary service compared to the baseline. Higher CPU consumption in the canary could point to inefficiencies in the new version.
- **Memory Usage**: Track memory consumption to ensure that the canary version does not exhibit memory leaks or excessive memory usage.
- **Disk and Network I/O**: If the service reads/writes to disk or communicates over the network, monitor disk I/O and network throughput for abnormalities.

# THROUGHPUT AND CAPACITY

- **Request Throughput**: Ensure that the canary can handle the expected volume of traffic without degrading its performance.
- **Capacity Saturation**: Look at the percentage of capacity the canary is consuming. If it is close to saturation, this could be a risk for a full-scale rollout.

# SERVICE-SPECIFIC METRICS

- **Custom Business KPIs**: If the microservices are tied to specific business outcomes (e.g., number of purchases, user sign-ups), monitor those key performance indicators (KPIs) to ensure that the canary is driving the same or better outcomes.
- **Internal Service Metrics**: Depending on the service, you may want to track things like cache hit rates, database query performance, or queue lengths for background jobs.

# AVAILABILITY AND HEALTH

- **Uptime/Availability**: Ensure that the canary version remains available throughout the test period. Sudden downtimes or frequent restarts can indicate underlying issues.
- **Health Checks**: Look at the results of automated health checks (e.g., liveness and readiness probes) to verify the canary is healthy and ready to serve traffic.

# USER BEHAVIOR/EXPERIENCE METRICS

- **User Interactions**: Monitor metrics that reflect how end-users are interacting with the service. Sudden drops in user engagement (e.g., fewer page visits or app interactions) can be a sign of issues with the canary version.
- **Error Logs**: Analyze logs for unusual patterns or a spike in user-facing errors.

# SERVICE DEPENDENCY METRICS

- **Downstream Service Latency and Errors**: If the canary service depends on other services, monitor how it interacts with them. Increased downstream errors or latency could signal issues in how the new version interacts with other parts of the system.
- **Third-Party API Call Success Rate**: If the microservice calls external APIs, monitor the success rate and latency of these API calls in the canary version.

# TRAFFIC DISTRIBUTION AND CANARY WEIGHT

- **Traffic Distribution**: Ensure that traffic is correctly distributed between the baseline and the canary according to the canary strategy (e.g., 5%, 10%, etc.). Unexpected spikes in canary traffic could indicate misconfiguration.
- **Canary Weight Adjustment**: Observe how increasing the percentage of traffic to the canary impacts the metrics. If no major issues arise as more traffic is routed to the canary, it is a positive signal for promotion.

# ALERTS AND ANOMALIES

- **Alerts** Set up alerts for key metrics like error rates, latency, and resource utilization. If the canary triggers fewer or no alerts compared to the baseline, it is a sign of stability.
- **Anomaly Detection**: Use Prometheus to detect anomalous behavior, such as sudden spikes in errors, latencies, or resource consumption.

# DEMO: INTEGRATING METRICS

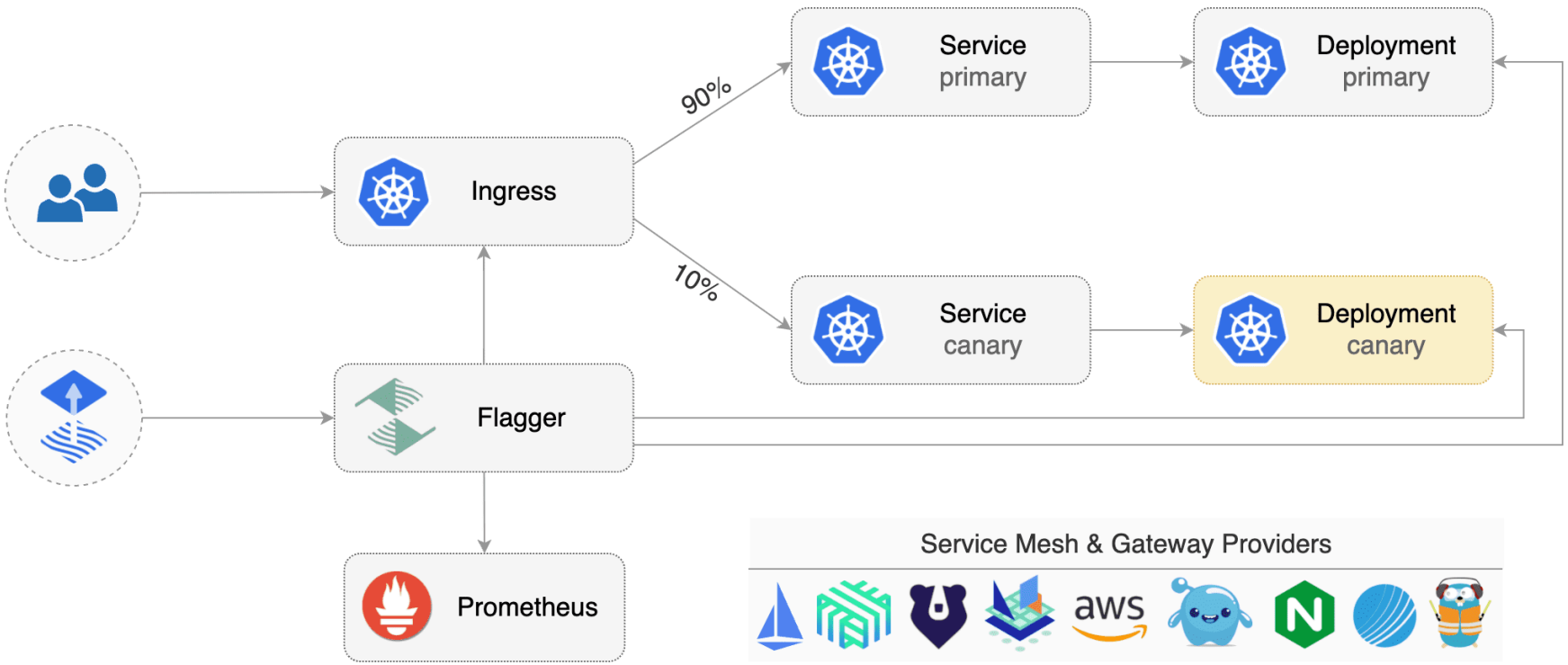
In the *canary-app* project, let's navigate to the *simple-microservice*, and run our simple Quarkus application that has metrics that we can use by Prometheus in our case.


FLAGGER

# WHAT IS FLAGGER?


- Flagger is a Kubernetes operator that automates the promotion or rollback of new software releases based on performance analysis.
- It's designed for progressive delivery, where new versions of applications are gradually rolled out in production.
- Works with service meshes like Istio, Linkerd, and ingress controllers like NGINX, Gloo, and Contour.

# FLAGGER OVERVIEW




Source: Flux CD & Flagger

# HOW IS IT RELATED TO ISTIO?

| Feature | Istio | Flagger |
|---|---|---|
| **Main Purpose** | Service Mesh and Network Traffic Control | Progressive Delivery (e.g., Canary, Blue/Green) |
| **Traffic Management** | Advanced traffic routing and resilience | Automated traffic shifting for deployments |
| **Health Monitoring** | Basic network-level metrics | Application-level health monitoring and rollback |
| **Deployment Strategy** | Manual or externally managed | Automated progressive delivery (canary, blue/green) |
| **Integration** | Integrates with observability/security | Integrates with Istio/NGINX and monitoring tools for automated decisions |

# FLAGGER KEY FEATURES

- Automates Canary and Blue/Green deployments.
- Tracks key performance metrics: Success Rate, Request Duration, and Error Rates.
- **Integrated with Prometheus, Datadog, and other monitoring tools to monitor app health**
- Provides automatic rollbacks in case of degraded performance or errors.

# FLAGGER DEPLOYMENT TYPES

**Flagger can run automated application analysis, testing, promotion and rollback for the following deployment strategies**

- Canary (progressive traffic shifting)
  - Istio , Linkerd , App Mesh , Open Service Mesh , Kuma Service Mesh
  - Contour , Gloo , NGINX , Skipper , Traefik
- A/B Testing (HTTP headers and cookies traffic routing)
  - Istio , App Mesh , Contour , NGINX
- Blue/Green (traffic switching and mirroring)
  - Kubernetes CNI (Container Network Interface), Istio, Linkerd, App Mesh, OSM, Kuma, Contour, Gloo, NGINX, Skipper, Traefik

# INSTALLATION OF FLAGGER

Flagger installation is dependent on the service mesh you are using, each with its own special set of instructions and tutorials

# CONFIGURING A CANARY RULE

- Typically, you will not require the all same yaml for services since it will take care of the rollout for you.
- You will specify what you want in a *canary.yaml* file where you will specify things like:
  - `maxweight`
  - `stepweight`
  - `interval`
  - Metrics to be used

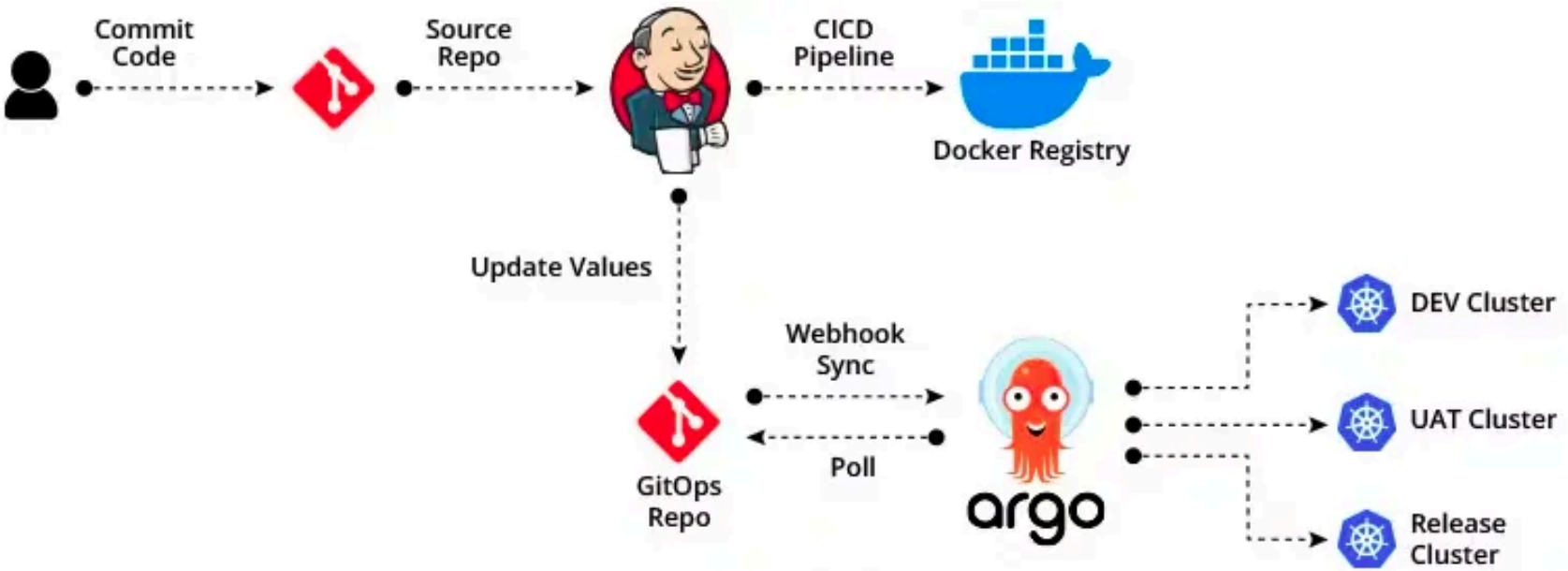
# HOW DO WE DO ROLLBACKS?

- Flagger continuously monitors its performance using key metrics such as success rate, request duration, and error rates.
- If any of these metrics fall below the acceptable thresholds, Flagger *automatically* initiates the rollback process.

# DEMO: FLAGGER AND ISTIO

In the *canary-app* project, let's navigate to the *flagger*. View the manifests and let's run the manifests


GITOPS




Everything about Argocd — Gitops

# EVERYTHING IS GIT

- In GitOps, everything is Git based
- It is *infrastructure as code*
- This includes:
  - Managing and Deployment Manifests
  - Rolling back to previous Manifests
  - Running tests and scans

# WHAT DO YOU PUT IN THESE REPOSITORIES?

- Terraform Files
- Ansible Scripts
- Kubernetes Deployments

# WHAT IS THE CHANGE PROCESS?

- Create a Pull/Merge request
- Collaborate with others
- Automatically run a CI/CD pipeline that will validate tests and scripts
- Approve the changes

# SEPARATE GIT REPOSITORY FROM THE APPLICATION

- It is standard practice to separate the application's Git repository from the configuration's Git repository
- In other words, one for the application, one for the configuration

# MOVING BACK IN HISTORY

- In order to roll back a particular change you can just either:
  - `git revert`
  - `get reset`
- This will be what you do to "rollback" to a previous version.

# CLUSTER DISASTER RECOVERY

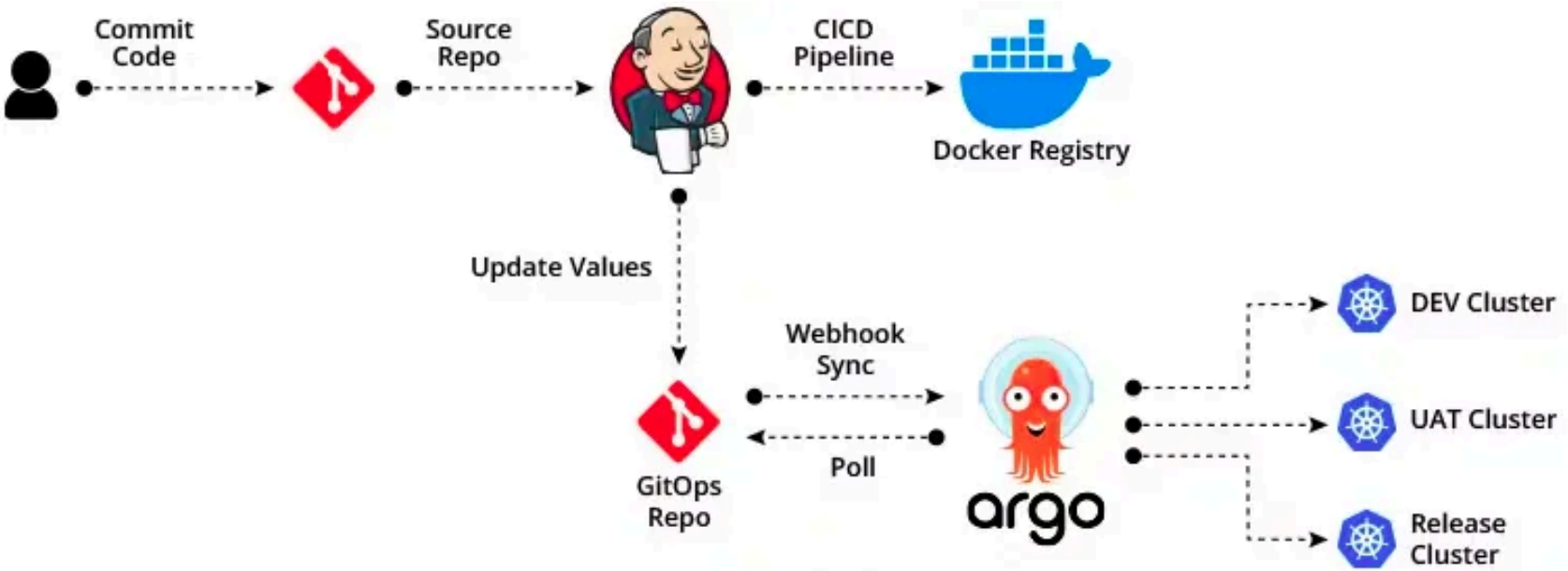
- When a cluster is destroyed for whatever reason, another can be created in its place
- We can direct the cluster to repository with the configuration

# PERMISSIONS THROUGH GIT

- All manifests are managed in a repository
- We can lock down access based on roles to that Git repository
- No requirement to create `ClusterRole` and `User` resources in Kubernetes

# NO NEED TO PROVIDE EXTERNAL CLUSTER ACCESS

- There is requirement to give external cluster access to non-human users
- No need to have cluster credentials outside of Kubernetes, the security is managed within Git.




Everything about Argocd — Gitops


ARGO CD

# WHAT IS ARGOCD

- Argo CD is a declarative, ***GitOps*** continuous delivery tool for Kubernetes.
- It automates the deployment of desired application states to Kubernetes.
- Argo CD monitors applications continuously and compares the live state with the desired state specified in a Git repository.
- It supports various Kubernetes cluster configurations and multiple clusters.

# WHAT IS ARGOCD ROLLOUTS?

- ArgoCD Rollouts is a Kubernetes controller and a set of CRDs (Custom Resource Definitions) that provide advanced deployment capabilities for Kubernetes applications.
- It supports features such as canary releases, blue-green deployments, and experimentation with different rollout strategies.
- Provides detailed deployment analysis and actionable insights, including real-time alerts and notifications.
- Offers traffic shaping and service mesh integrations to control the flow of traffic during deployments.
- Ensures smooth upgrades and rollbacks with minimal disruption and maximum reliability.

# HOW ARE ARGOCD ROLLBACKS PERFORMED?

- ArgoCD Rollbacks are performed by reverting to a previous application state defined in the Git repository.
- The system uses the Git commit history to identify the specific version to which to rollback.
- The desired state specified in the selected Git commit is applied to the Kubernetes cluster, reversing any changes made by subsequent deployments.
- ArgoCD continuously monitors the rollback process to ensure the cluster reaches the desired state.

# DEMO: ARGO CD

In the *canary-app* project and the *canary-app-config* and view the manifest, and we will run them to view ArgoCD and see canaries in Action


CANARY DEPLOYMENTS IN SPINNAKER

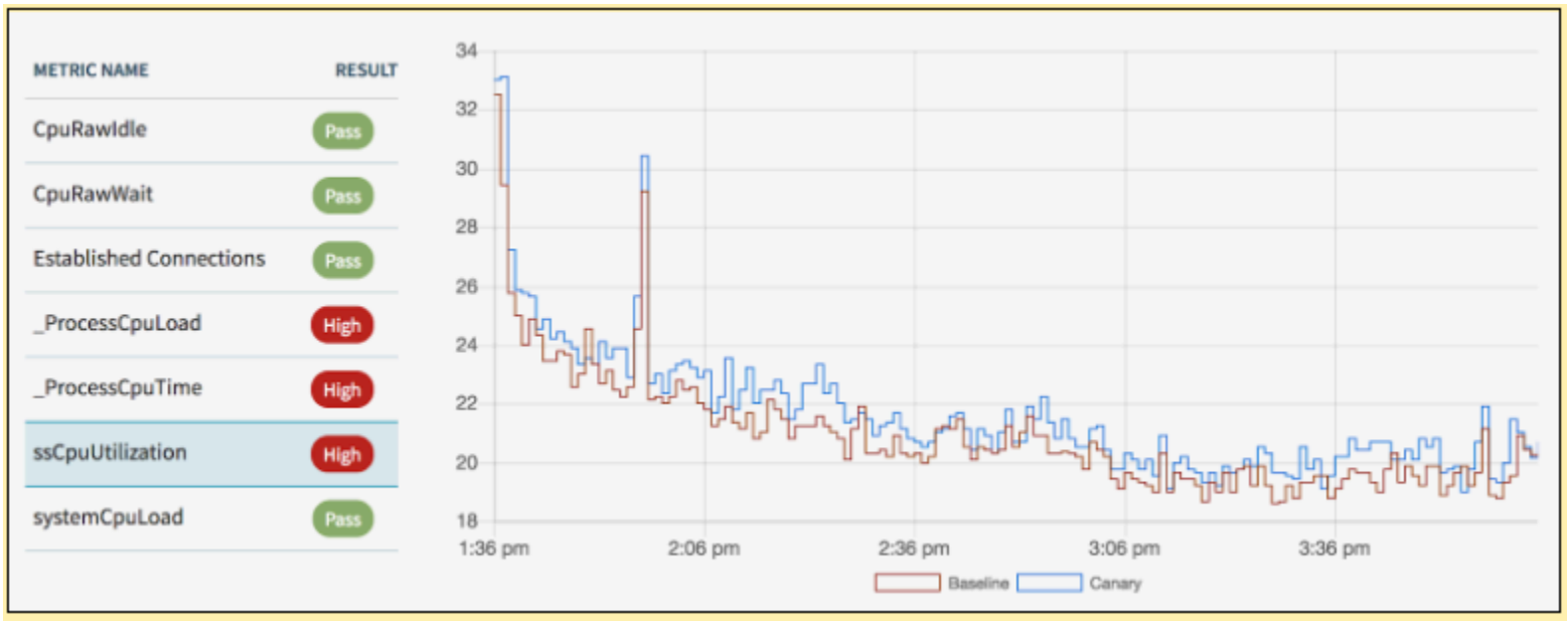
# ABOUT SPINNAKER


- Spinnaker is an open-source continuous delivery platform originally developed by Netflix.
- It supports multi-cloud deployments, allowing teams to deploy applications across public, private, and hybrid cloud environments.
- Spinnaker integrates with various cloud providers like AWS, Google Cloud, and Kubernetes, among others.
- It provides automated canary analysis, which helps in incrementally releasing changes to a small subset of users and monitoring metrics before a full rollout.
- Spinnaker's pipeline feature allows for sophisticated, multistep release processes, encouraging best practices in deployment automation.

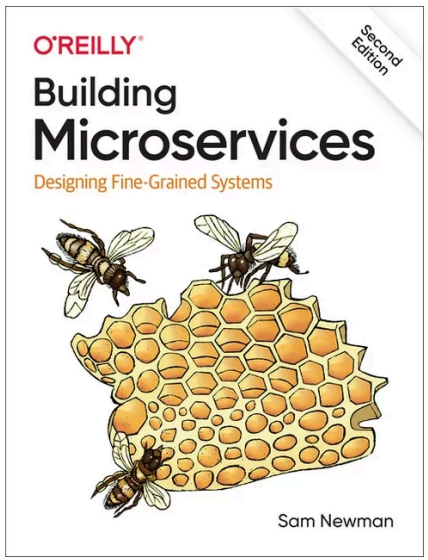
# SPINNAKER AND CANARY

- Canary deployment in Spinnaker allows teams to incrementally release changes to a small subset of users before performing a full rollout.
- The process begins by deploying a new version of the application to a small percentage of the total instances, known as the canary.
- Traffic is then routed to the canary in a controlled manner, while the rest of the traffic continues to go to the stable version.
- Performance and health metrics of the canary are monitored and compared against the baseline (i.e., the stable version).
- If the canary performs within acceptable thresholds, its deployment gradually increases to a larger percentage of instances.
- If the canary fails to meet the required metrics, the changes can be rolled back to prevent widespread issues.
- Spinnaker uses the **Kayenta** microservice for automated canary analysis (ACA).
- Kayenta integrates with various metrics services (e.g., Prometheus) to evaluate and score the performance of canary deployments.

# SPINNAKER KAYENTA

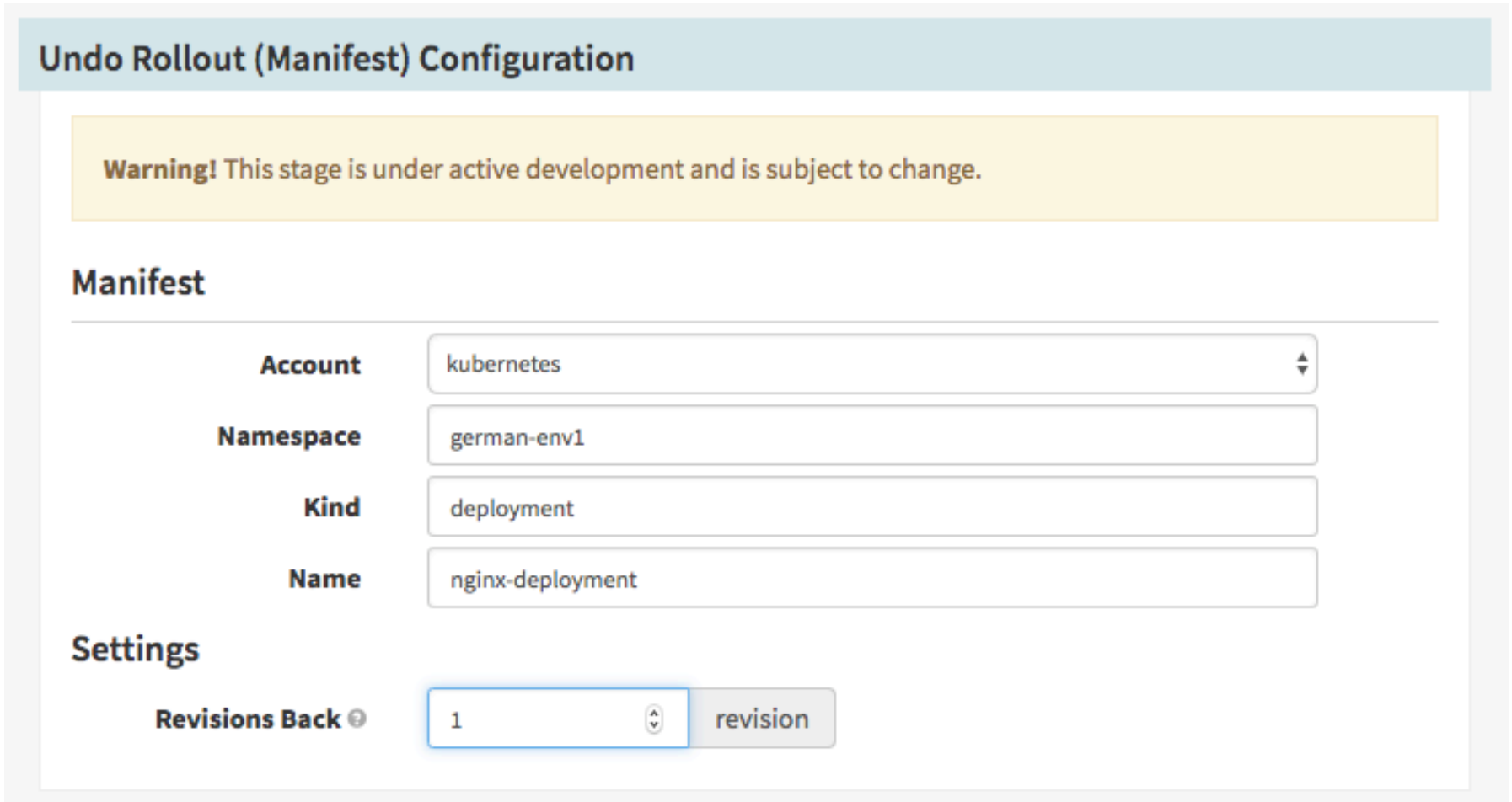


# SAM NEWMAN QUOTE ON SPINNAKER




*When I first did a canary release we controlled the rollout manually. We could configure the percentage of our traffic seeing the new functionality, and over a period of a week we gradually increased this until everyone saw the new functionality. Over the week, we kept an eye on our error rates, bug reports, and the like. Nowadays, it's more common to see this process handled in an automated fashion. Tools like Spinnaker for example have the ability to automatically ramp up calls based on metrics, such as increasing the percentage of calls to a new microservice version if the error rates are at an acceptable level.*

# HOW DO WE DO ROLLBACKS IN SPINNAKER?




AWS AND CANARY DEPLOYMENTS

# KUBERNETES AND AWS PARALLELS

| Kubernetes Concept | AWS Equivalent |
| --- | --- |
| Pod | EC2 Instance (Virtual Machine) |
| Node | EC2 Instance (Host for running Pods) |
| Deployment | Auto Scaling Group (Managing lifecycle and scaling of instances) |
| ReplicaSet | Auto Scaling Group's Instance Set (Ensuring the correct number of instances are running) |
| Service (ClusterIP) | Internal Load Balancer (Distributing traffic within the VPC) |
| Service (NodePort) | EC2 Security Group with Open Ports (External access to services) |

# KUBERNETES AND AWS PARALLELS (CONTINUED)

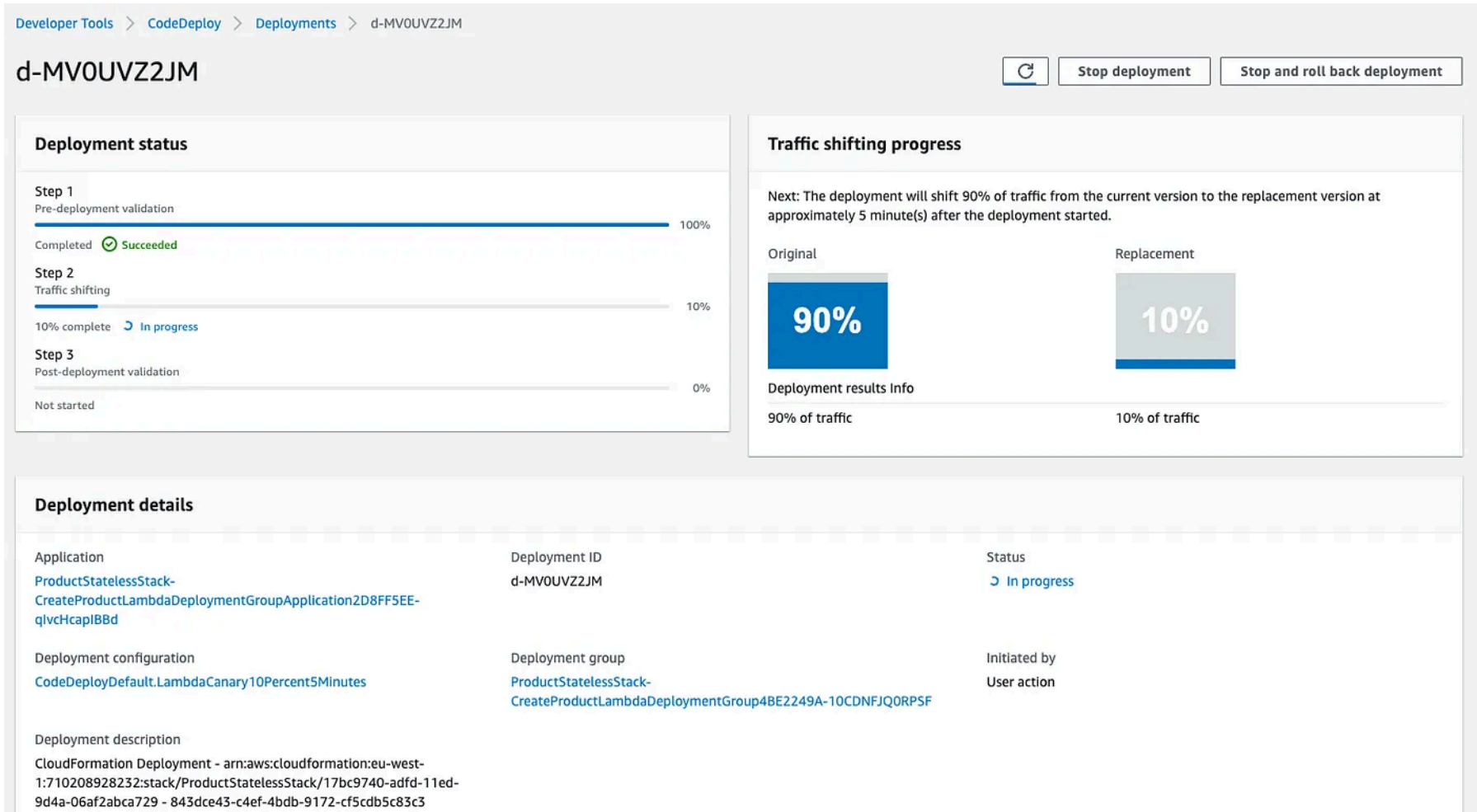
| Kubernetes Concept | AWS Equivalent |
| --- | --- |
| Service (LoadBalancer) | Elastic Load Balancer (ELB) / Application Load Balancer (ALB) (External load balancer for traffic) |
| Ingress | API Gateway (Managing external access and routing) |
| ConfigMap / Secret | Parameter Store / Secrets Manager (Storing configuration data and secrets) |
| Persistent Volume (PV) | EBS Volume (Providing storage for instances) |
| Persistent Volume Claim (PVC) | EBS Volume Attachment (Binding storage to a specific instance) |
| Namespace | AWS Account / VPC (Logical isolation of resources) |

# USING AWS CODE DEPLOY

- In 2017, AWS added functionality to Lambda Aliases allowing traffic shifting of lambda function versions based on pre-assigned weights.
- This functionality allows traffic to be gradually shifted from one lambda version to another over a period of time based on a predefined percentage.

Source: Lambda Canary Deployments with Code Deploy Using the CDK

# USING AWS CODE DEPLOY




Source: Lambda Canary Deployments with Code Deploy Using the CDK


CONCLUSION

# CANARY BENEFITS

- Risk Mitigation
- Early Detection of Issues
- Improved User Experience
- Performance Monitoring
- Innovation and Experimentation
- Feedback Loop
- Rollback Strategy

# THANK YOU

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