#argocon

ArgoCon EUROPE

Automated Resilience:

Using Argo Events for Real-Time Incident Remediation

Darko Janjić – Pipekit

About Pipekit

Scale Argo & Kubernetes with Pipekit

- Direct support from 40% of the active Argo Workflows maintainers in the world.
- Save engineering time and up to 60% on compute costs
- Add 3 Argo maintainers and 7 Argo contributors to your team
- Serving startups & Fortune 500 enterprises since 2021:

Enterprise Support for Argo:
Ideal for Platform Eng teams scaling with Argo

Control Plane for Argo Workflows:
Ideal for data teams, granular RBAC, and multi-cluster architectures

What is Auto-remediation?

- Process of detecting and resolving system failures automatically
- Enables self-healing of applications/infrastructure
- Integrates with development tools, security platforms, and CI/CD pipelines to provide real-time or near-real-time responses to detected issues.

How Auto-remediation works?

Detection

- Monitoring tools that continuously track system
- Triggers alerts upon detected issues
- Can be AI powered

Decision: Event processing and Rule-Based Automation

- Receives a notification from detection engine
- Determines appropriate action
- Triggers action

Argo Events

Amazon EventBridge

RabbitMQ

Action: Automated Remediation Workflows

- Receives a command from detection engine
- Determines appropriate action
- Triggers action

Argo Workflows

ArgoCD

Auto-remediation (recap)

- Process of detecting and resolving system failures automatically
- Detection → Decision → Action
- Minimizes downtime, reduces operational costs, and enhances security and compliance

What is Argo Events?

Argo Workflows

Kubernetes-native workflow engine supporting DAG and step-based workflows

Argo CD

Declarative continuous delivery with a fully-loaded UI

Argo Events

Event based dependency management for Kubernetes.

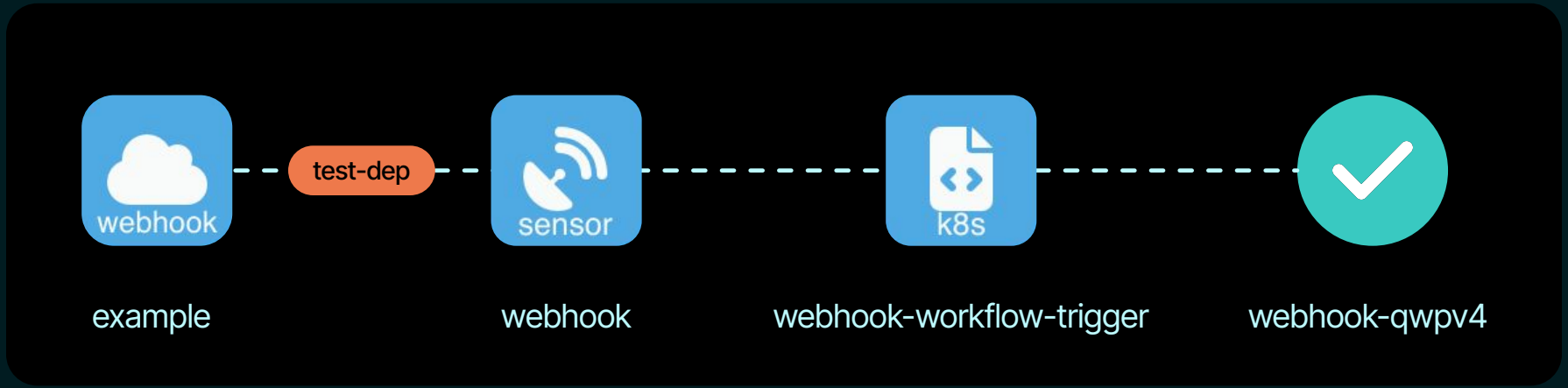
Argo Rollouts

Advanced Kubernetes deployment strategies such as Canary and Blue-Green made easy.

What is Argo Events? (definition)

Argo Events is an event-driven workflow automation framework.

The Argo Events UI lives within Argo Workflows



What is Argo Events? (use cases)

CI/CD Pipelines

Event-based data processing

General Automation

ArgoCon EUROPE

What is Argo Events? (components)

Demystifying Argo Events talk

4
COMPONENTS

Sensor

Defines the connection between the Event Source and the resource being triggered

Event Source

Describes configuration for consuming events from external sources

Controller Manager

Observes creation/changes to Event Source, Event Bus and Sensor custom resources.

Event Bus

A transport layer that connects the Event Source to the Sensor

Demo

- Simulate crypto miner threat and notify Argo Events
- Argo Events filters the events and triggers the high severity ones
- Workflow is executed
- Capture affected Pod metadata and env vars
- Isolate the pod using NetworkPolicy
- Scan Container image using Trivy
- Alert using Slack
- Cordon node

Demo - Event Source

```yaml
apiVersion: argoproj.io/v1alpha1
kind: EventSource
metadata:
 name: webhook
spec:
 service:
   ports:
     - port: 12000
       targetPort: 12000
 webhook:
   notify:
     port: "12000"
     endpoint: /notify
     method: POST
     filter:
       expression: "body.status == 'firing' &&
body.commonLabels.severity == 'critical' &&
body.commonLabels.alertname == 'CryptoMinerDetected'"
     authSecret:
       name: webhook-secret
       key: auth
```

Demo - Sensor

```yaml
spec:
 template:
   serviceAccountName: operate-workflow-sa
 dependencies:
   - name: webhook-dep
     eventSourceName: webhook
     eventName: notify
     transform:
       script: |-
         local json = {
           pod = event.body.alerts[1].labels.pod,
           namespace = event.body.alerts[1].labels.namespace,
           instance = event.body.alerts[1].labels.instance,
           container = event.body.alerts[1].labels.container
         }
         event.body = json
         return event
```

```yaml
triggers:
 - template:
     name: webhook-workflow-trigger
     k8s:
       operation: create
       source: ...
       parameters:
         - src:
             dependencyName: webhook-dep
             dataTemplate: "{{ .Input.body.pod }}"
           dest: spec.arguments.parameters.0.value
         - src:
             dependencyName: webhook-dep
             dataTemplate: "{{ .Input.body.namespace }}"
           dest: spec.arguments.parameters.1.value
         - src:
             dependencyName: webhook-dep
             dataTemplate: "{{ .Input.body.instance }}"
           dest: spec.arguments.parameters.2.value
         - src:
             dependencyName: webhook-dep
             dataTemplate: "{{ .Input.body.container }}"
           dest: spec.arguments.parameters.3.value
```

ArgoCon EUROPE

Demo - Workflow

```yaml
spec:
 entrypoint: main
 arguments:
   parameters:
     - name: pod
     - name: namespace
     - name: instance
     - name: container
 templates:
   - name: main
     steps:
       - - name: capture-pod-data
           template: capture-data
       - - name: isolate-pod
           template: isolate
       - - name: scan-image
           template: image-scan
           arguments:
             parameters:
               - name: image
                 value: "{{steps.capture-pod-data.outputs.parameters.image}}"
       - - name: notify-team
           template: notify
       - - name: escalate
           template: escalate
           when: "{{steps.scan-image.status}} != Succeeded"
       - - name: cordon-node
           template: cordon-node
```

Demo - capture-data step

```
- name: capture-data
  container:
    image: bitnami/kubectl
    command: [ "sh", "-c" ]
    args:
      - |
        kubectl -n {{workflow.parameters.namespace}} logs {{workflow.parameters.pod}} > /tmp/pod-logs
        kubectl -n {{workflow.parameters.namespace}} describe pod {{workflow.parameters.pod}} > /tmp/pod-description
        kubectl -n {{workflow.parameters.namespace}} get pod {{workflow.parameters.pod}} -o
jsonpath='{.spec.containers[?(@.name=="{{workflow.parameters.container}}")].image}' > /tmp/image
        echo "Pod data captured."
  outputs:
    artifacts:
      - name: pod-logs
        path: /tmp/pod-logs
        s3:
          key: "{{workflow.parameters.pod}}-logs"
      - name: pod-description
        path: /tmp/pod-description
        s3:
          key: "{{workflow.parameters.pod}}-description"
    parameters:
      - name: image
        valueFrom:
          path: /tmp/image
```

ArgoCon EUROPE

Demo - isolate step

```
- name: isolate
  resource:
    action: create
    manifest: |
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: isolate-{{workflow.parameters.pod}}
        namespace: {{workflow.parameters.namespace}}
      spec:
        podSelector:
          matchLabels:
            name: {{workflow.parameters.pod}}
        policyTypes:
        - Ingress
        - Egress
        ingress: []
        egress: []
```

Demo - image-scan step

```
- name: image-scan
 inputs:
   parameters:
     - name: image
 container:
   image: aquasec/trivy
   command: [ "trivy" ]
   args: [ "image",
"{{inputs.parameters.image}}" ]
```

Demo - notify step

```
- name: notify
  script:
    image: ghcr.io/tico24/whalesay
    command: [ cowsay ]
    args: ["Notifying users of
suspicious pod
{{workflow.parameters.pod}}."]
```

ArgoCon EUROPE

Demo - cordon-node step

```
- name: cordon-node
  resource:
    action: patch
    manifest: |
      apiVersion: v1
      kind: Node
      metadata:
        name:
{{workflow.parameters.instance}}
      spec:
        unschedulable: true
```

Demo - Alert Manager payload

```
"status": "firing",
"alerts": [
 {
   "status": "firing",
   "labels": {
     "alertname": "CryptoMinerDetected",
     "severity": "critical",
     "instance": "docker-desktop",
     "job": "kubernetes-nodes",
     "namespace": "default",
     "pod": "worker",
     "container": "ubuntu",
     "reason": "High_CPU_Usage",
     "threshold": "90"
   }
]
...
```

Demo

- Argo Events received the notification
- Argo Events filtered and decided which notifications are important
- Started remediation workflow
- Workflow captured affected pod metadata and env vars
- Isolated the pod using NetworkPolicy
- Scan container image using Trivy
- Alerted on-call person using Slack
- Cordoned the node

RabbitMQ as events handler/decision maker

- Not a native Kubernetes app
- Custom producer needed to receive the event, validate it and pass it to the queue
- Custom consumer needed to handle the events
- Not lightweight
- Not easy to scale

Custom app as events handler/decision maker

- Time needed to develop and support
- Need to implement every new event source
- No monitoring out of the box
- No built-in fault tolerance
- Manual scaling
- Extra work to secure
- Higher maintenance cost

Why Argo Events?

- Cloud native and container native
- Open-source
- Easy to filter and decide which events you want to process
- Plug and play
- Scalable
- Easy to change and update
- Many out of the box inputs
- Many out of the box outputs

Best practices

- Validate event
- Filter events
- Add retry policies and DLQ
- Make operations idempotent
- Alerting
- Avoid event loops
- Test

Why would you use this?

- Less prone to error
- Can act faster than the on-call person
- Minimizes downtime and service disruptions
- Improves reliability with self-healing infrastructure
- Reduces operational costs and manual work
- Auto-remediates threats and enforces policies.

Recap

- Argo Events
- Auto-remediation
- Demo
- Best practices and benefits

Stuff!

This and previous Pipekit talks:

https://pipek.it/talk-demos

Free Argo/Infrastructure Help & Advice:

Booth S590 - Pipekit Booth

Free Argo resources by Pipekit