Overview
WCNP - KiTT
&
Best Practices

Agenda

- WCNP Overview & Adoption
- KITT Native and Offerings
  - Build & Deployment
  - Profiles
  - Autoscaling
  - Hooks & Automated Tests
  - Release Strategies
- Best Practices
- Monitoring & Alerting

WCNP

The Walmart Cloud Native Platform

Kubernetes based platform that securely and efficiently runs Walmart's cloud native, containerized workloads.

Definitions

- **Container:** decouple applications from underlying host infrastructure.
- **POD:** group of one or more containers in k8s.
- **Node:** worker machine in k8s cluster hosts PODs.
- **Cluster:** a set of nodes (worker machines).
- **Namespace:** virtual clusters backed by the same physical cluster.

App Requirements

The twelve-factor app

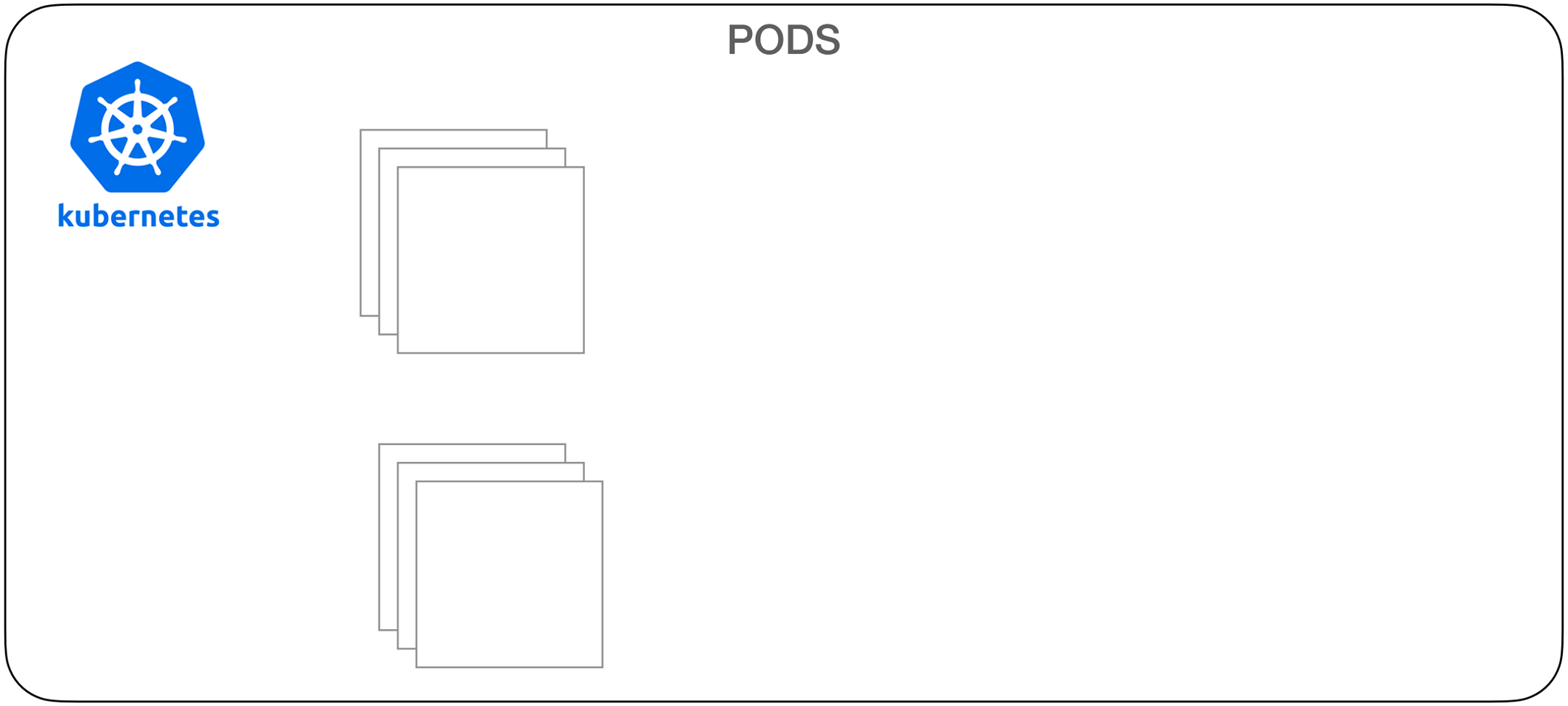
- **Codebase**: One codebase tracked in revision control, many deploys
- **Dependencies**: Explicitly declare and isolate dependencies
- **Config**: Store config in the environment
- **Backing services**: Treat backing services as attached resources
- **Build, release, run**: Strictly separate build and run stages
- **Processes**: Execute the app as one or more stateless processes and share-nothing.
- **Port binding**: Export services via port binding

App Requirements

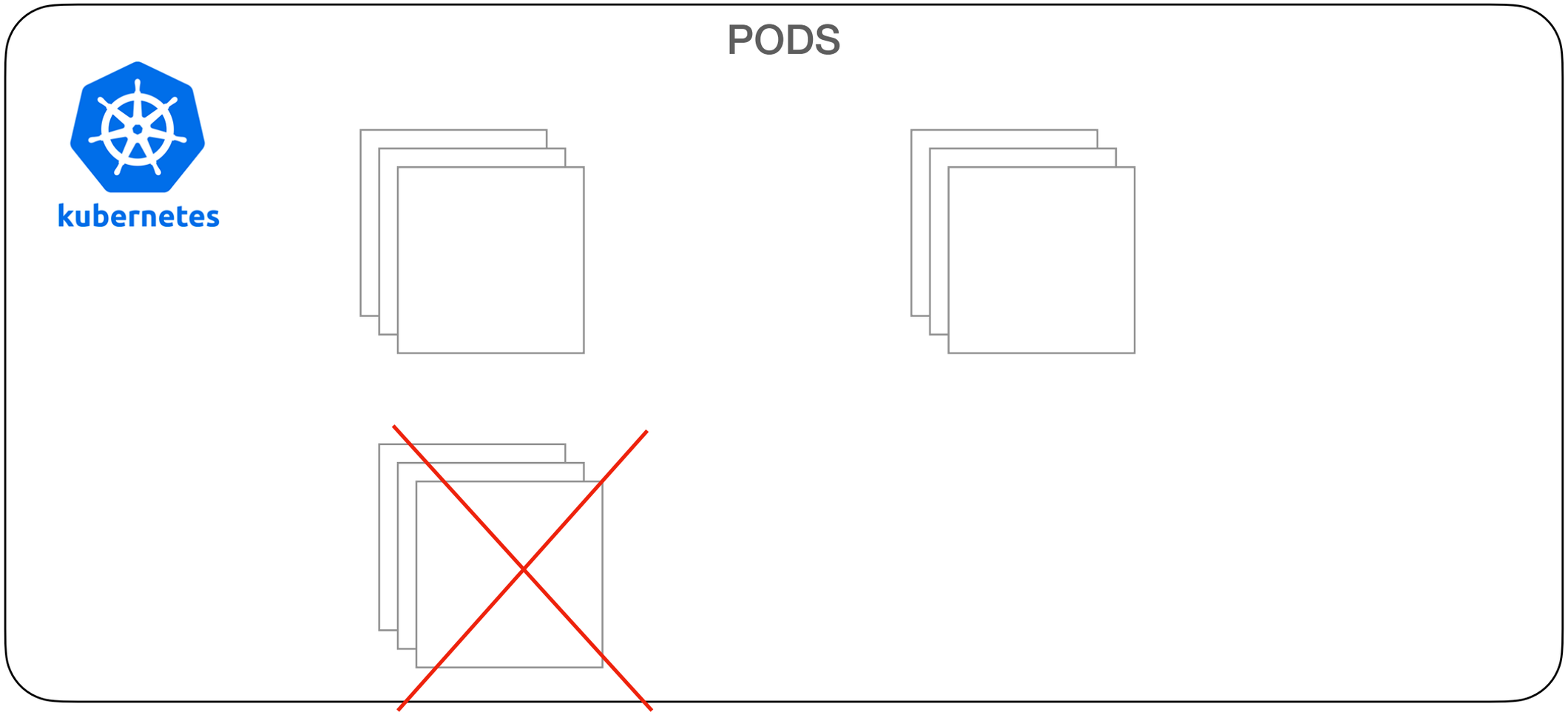
The twelve-factor app

- **Concurrency:** Scale out via the process model
- **Disposability:** Maximize robustness with fast startup and graceful shutdown
- **Dev/Prod parity**: Keep development, staging, and production as similar as possible
- **Logs:** Treat logs as event streams
- **Admin processes:** Run admin/management tasks as one-off processes

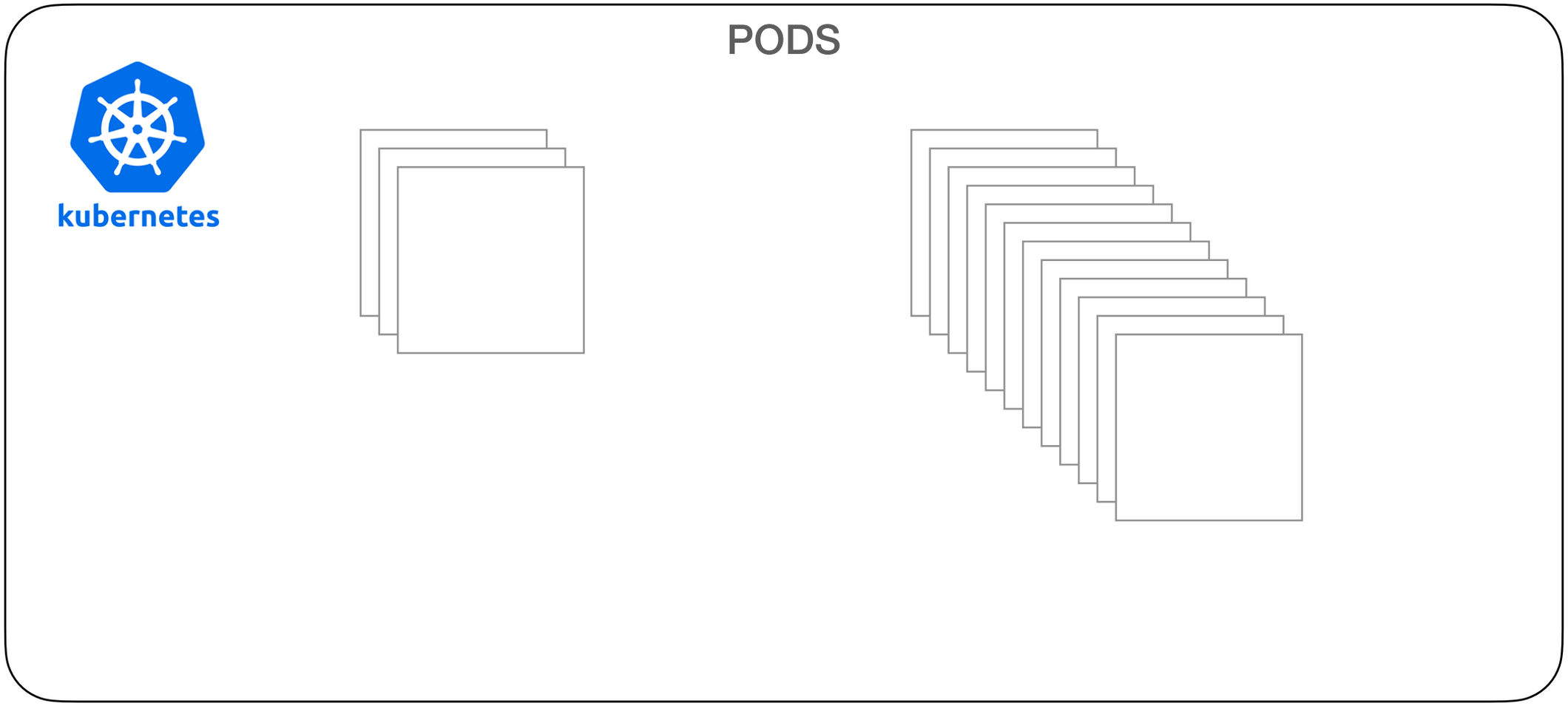
WCNP



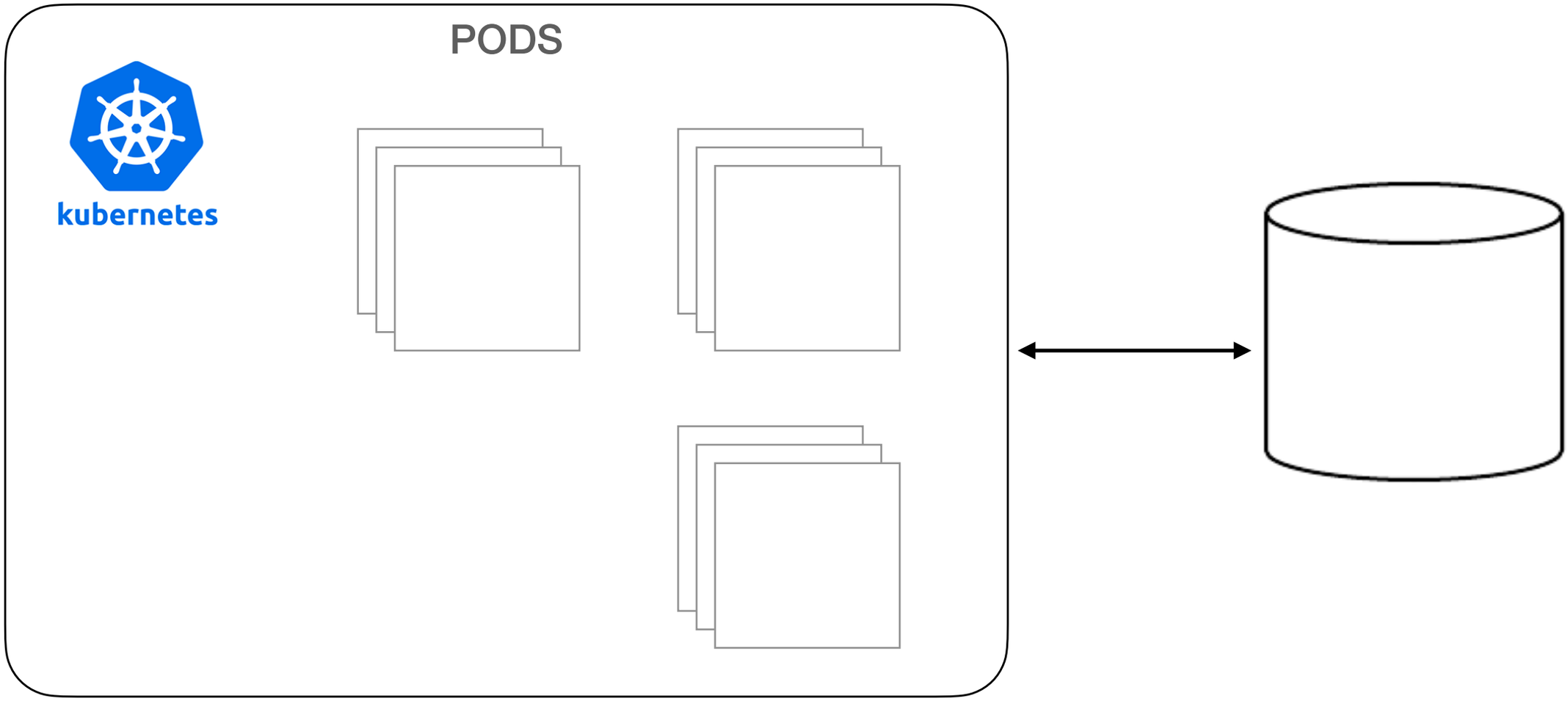
WCNP



WCNP



WCNP



WCNP

Storage: Strati Managed Service

- Azure BLOB Storage
- CosmosDB
- Cassandra
- MegaCache
- more....

https://strati.walmart.com/products/index.html#databases

WCNP

Adoptions

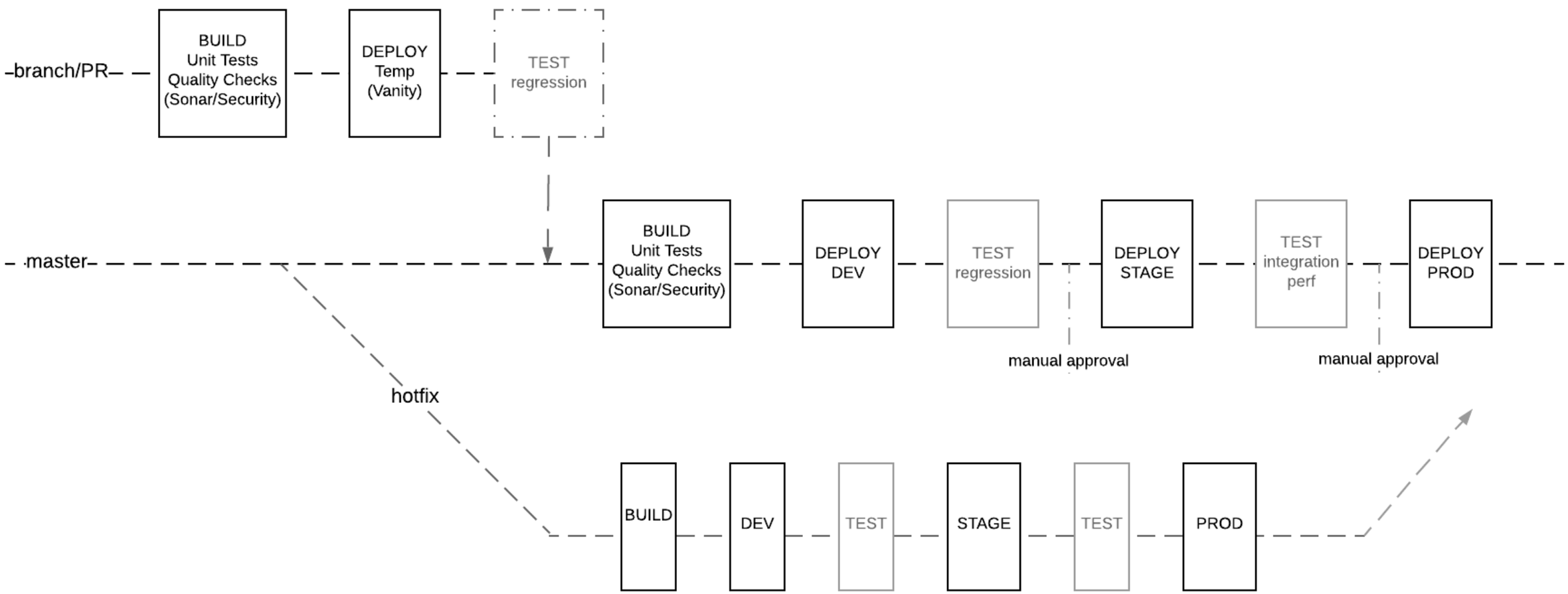
- Kitt Native
- Kitt + Custom Docker
- BYOCI

Kitt Native

- Full pipeline from build to monitoring and reporting
- Auto generated CICD (Looper/Concord).
- Can co-exists with current CICD
- Trunk based development - best practice
- Release branches are supported

Kitt Native

Trunk based Development



kitt.yml

```yaml
profiles:
  - tomcat7-jdk8-centos

owner:
  group: AD-GROUP-NAME

notify:
  slack:
    channelName: <slack-channel-name>

deploy:
  namespace: "your-namespace"
  releaseType:
    deployTimeout: 600
```

kitt.yml

```yaml
stages:
  - name: dev
    flows: [release, pr, branch]
    target:
      - cluster_id: [scus-dev-a2]

  - name: stage
    flows: [release]
    target:
      - cluster_id: [scus-stage-a3]

  - name: prod
    flows: [release]
    target:
      - cluster_id: [wus-prod-a4]
```

- **name** - how it will be identified throughout the flow
- **flows** - what will trigger this build stage
- **target** - what cluster this will deploy to

kitt.yml

```
stages:
  . . . .
  - name: stage
    approvers:
      groups:
        - AD-GROUP-NAME
    flows: [release]
    target:
      - cluster_id: [scus-stage-a3]
  - name: prod
    approvers:
      groups:
        - AD-GROUP-NAME
    flows: [release]
    target:
      - cluster_id: [wus-prod-a4]
```

Adding approvals

Kitt Native

Branch/PR Deployments

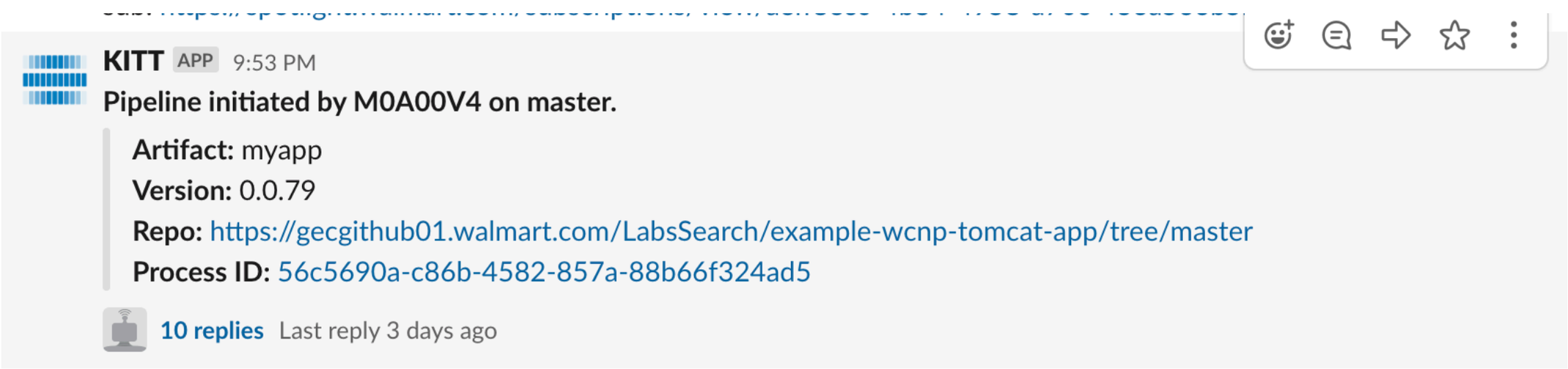
- Each branch and PR will be built and deployed to a unique temporary namespace with a unique url for your commit/PR only.
- No need for vanity environments.
- You can see the url in your build's concord logs as follows:

```
kube: You can access it via any of the below endpoints:
kube:  http://myapp.n133077760.scus-dev-a2.cluster.k8s.us.walmart.net
```

Kitt Native

Monitoring Pipeline

Can see it all in your Slack channel!



Kitt Native

Monitoring Pipeline

Thread
#example-wcnp-tomcat-app

KITT APP Mar 27th at 9:55 PM
Pipeline initiated by M0A00V4 on master.

> **Artifact:** myapp
> **Version:** 0.0.80
> **Repo:** https://gecgithub01.walmart.com/LabsSearch/example-wcnp-tomcat-app/tree/master
> **Process ID:** d0103884-50ed-4621-be23-96dde75a2df1

13 replies

KITT APP 6 days ago
Build started on master.

KITT APP 6 days ago
Build was successful, resuming any deployment steps.

KITT APP 6 days ago
Beginning deployment to dev

KITT APP 6 days ago
Deployment to dev successful.

KITT APP 6 days ago
Promotion Needs Approval

> **Approval Form:** 'https://concord.prod.walmart.com/#/process/d0103884-50ed-4621-be23-96dde75a2df1/wizard'

KITT APP 2 days ago
Promotion was approved by m0a00v4

KITT APP 2 days ago
Beginning deployment to stage

KITT APP 2 days ago
Deployment to stage successful.

Kitt Native

Monitoring Pipeline

strati / pipeline / **d0103884-50ed-4621-be23-96dde75a2df1** FINISHED

Status | Events | Ansible | Logs | History | Wait Conditions | Child Processes | Attachments

Parent ID	-
Initiator	m0a00v4
Type	Default
Created At	2020-03-27 21:54:37
Start At	-
Last Update	2020-03-31 10:14:45
Timeout	~ 2h

Project	pipeline
Concord Repository	kitt.master
Repository URL	https://gecgithub01.walmart.com/strati/kitt.git
Repository Path	-
Commit ID	ef7b925b758fc705142953deb987a2ba4b14253a
Process Tags	-
Triggered By	GitHub

CHECKPOINTS

Run #1

start | build-step | deploy | stage dev | stage stage | stage prod

kitt.yml

Liveness/Readiness Probes

- **livenessProbe:** K8s uses liveness probes to know when to restart a container.
- **readinessProbe:** Indicates whether the Container is ready to service requests.
- K8s doesn't guarantee Zero Downtime rollouts out-of-the-box
- unless readiness endpoint correctly configured

```
deploy:
  helm:
    values:
      livenessProbe:
        enabled: "true"
        probeInterval: 30
        path: "/app/status"
      readinessProbe:
        enabled: "true"
        probeInterval: 30
        path: "/app/status"
```

kitt.yml

Resource Configuration

CPU, memory, and scaling requirements for your workload.

```yml
stages:
  - name: prod
    flows: [release]
    target:
      - cluster_id: [scus-prod-a2]
    helm:
      values:
         min:
           cpu: 1
           memory: 512Mi
         max:
           cpu: 2
           memory: 2048Mi
```

kitt.yml

Resource Configuration - Cost Calculation

```
 min:
   cpu: 1
   memory: 512Mi
max:
   cpu: 2
   memory: 2048Mi
```

min (request value): the min guaranteed.
max (limit value): if available on that node.

- On an hourly basis, Cost Analyzer max (request value / actual utilization).
- If you don’t burst up to your limit, you do not pay for that limit.

kitt.yml

Autoscaling

Kubernetes can automatically scale your application based on metrics queries you specify.

CPU Scaling

```
scaling:
  enabled: "true"
  cpuPercent: 80
  max: 10
  min: 2
```

Istio/envoy Metrics

```
profiles:
  - custom-scale-istio-http-requests
```

(*) provides scaling at targetAverageValue: 1000

kitt.yml

Autoscaling

Application Specific Metrics

```
deploy:
  helm:
    values:
      scaling:
        custom: true
        enabled: true
         min: 1
         max: 3
        prometheusQueries:
          http-events-per-second:
            queryContent: sum(rate(http_requests_seconds[2m]))
            targetAverageValue: 1000
```

kitt.yml

Autoscaling

Scheduled: used with custom metric based scaling

```
profiles:
  - cron-scale

deploy:
  helm:
    values:
      cronscale:
        scaleUp:
          schedule: "*/5 * * * *"
          min: "2"
          max: "6"
        scaleDown:
          schedule: "*/4 * * * *"
          min: “1"
          max: "3"
```

kitt.yml

Security Checks

```
git://LabsSearch:kitt-profiles:master:sonar
```

```
build:
 docker:
   app:
     buildArgs:
       sonarOpts:
           "-Dsonar.projectVersion={{ $.kitt.build.version }}
            -Dsonar.java.binaries=target/classes
            -Dsonar.junit.reportPaths=target/surefire-reports"
```

kitt.yml

Profiles

```
profiles:
  - tomcat7-jdk8-centos
  - git://LabsSearch:kitt-profiles:master:tomcat-conf
  - git://LabsSearch:kitt-profiles:master:sonar
  - ./metrics.yml
  - ./perf-test.yml
```

kitt.yml

Profiles

```
profiles:
  - higher-precedence-profile
  - lower-precedence-profile
  ...
```

kitt.yml

Hooks

```
deploy:
  preDeploy:
  - concord:
      name: basicconcord
      action: start
      org: Default
      project: examples
      repo: hello_world
```

```
deploy:
  postDeploy:
  - concord:
      name: basicconcord
      action: start
      org: Default
      project: examples
      repo: hello_world
```

kitt.yml

Hooks

```
deploy:
  preDeploy:
  - job:
      name: basicjob
      action: create
      timeoutSeconds: 120
      pollPeriodSeconds: 10
      namespace: mynamespace
      namePrefix: mytestjob
      image: docker.prod.walmart.com/library/alpine:3.8
      env:
        name1: value1
        name2: value2
        hey: ho
      command: printenv
```

kitt.yml

Hooks

```
deploy:
  postDeploy:
    - job:
        name: basicjob
        action: create
        timeoutSeconds: 120
        pollPeriodSeconds: 10
        namespace: mynamespace
        namePrefix: mytestjob
        image: hub.docker.prod.walmart.com/library/alpine:3.8
        env:
          name1: value1
          name2: value2
       command: printenv
```

kitt.yml

Automated Testing

- Use hooks to run tests in a a pre built container.
- Build your test container with application

```
build:
  buildType: "maven"
  artifact: "myapp"
  docker:
    app:
      dockerFile: "Dockerfile"
    tests:
      dockerFile: Dockerfile
      contextDir: ./tests/
```

kitt.yml

Regression Tests

- Tests will be stored in service repository with the code (`regwork` repository is also supported).
- Versioned and tagged with the code.
- Kitt will build a test container:

  service container : `docker.walmart.com/myapp:x.y.z`

  test container : `docker.walmart.com/myapp-tests:x.y.z`
- Run for PR/Branches (optional)
- Roll back will trigger the tests for the version you are rolling back to.

kitt.yml

Automated Testing

Example in-repo-compiled, cluster-level test execution hook.

```
postDeploy:
  - job:
      action: create
      name: "{{$.kitt.build.artifact}}-tests"
      timeoutSeconds: 300
      pollPeriodSeconds: 30
      fetchJobLogsAfterPolling: true
      namespace: "{{$.kitt.deploy.namespace}}"
      namePrefix: "{{$.kitt.build.artifact}}"
      image: "{{$.kitt.build.docker.tests.image}}:{{$.kitt.build.version}}"
      env:
        vip: "{{$.kittExec.currentCluster.hosts[0]}}"
        priority: "p1,p2,p3"
```

kitt.yml

Regression Tests

```
[=-=-=-=-=-=-=-=-=-=-=-= FAILED : [0/4] =-=-=-=-=-=-=-=-=-=-=-=]

+---+---------+------+-------+------+----------+---------+-------+
| # | Feature | Test | Query | Host | Priority | Message | Owner |
+---+---------+------+-------+------+----------+---------+-------+
+---+---------+------+-------+------+----------+---------+-------+

[=-=-=-=-=-=-=-=-=-=-=-= PASSED : [4/4] =-=-=-=-=-=-=-=-=-=-=-=]

]
[2020-04-07 21:31:44,776][INFO][#regwork.py][L:336]:[Status: PASSED]
[2020-04-07 21:31:44,777][INFO][#summarizer.py][L:97]:[Host Performance Metrics: {
  "typeahead-gm.dev.walmartlabs.com": {
    "fail": 0,
    "pass": 4
  }
}]
[2020-04-07 21:31:44,777][INFO][#regwork.py][L:445]:[Cronus environment and/or job url not provided. Will not
[2020-04-07 21:31:44,777][INFO][#regwork.py][L:120]:[Regression Framework run complete]
[2020-04-07 21:31:44,777][INFO][#run_regwork.py][L:44]:[Total Run time : 1 secs.]
[2020-04-07 21:31:44,777][INFO][#run_regwork.py][L:46]:[+++++++++++ Check Regression Result  +++++++++++++]
[2020-04-07 21:31:44,777][INFO][#run_regwork.py][L:63]:[#----------------END----------------#]
```

```
17:32:04 [INFO ] c.w.concord.plugins.log.LoggingTask - ......Executed postDeploy process-hooks with target clu
```

kitt.yml

Service Now Change Record

```
deploy:
  createChangeRecord: false
```

- ServiceNow record is created when the cluster deployment begins, and is closed automatically with a status of success for successful deployments, success with issues if a rollback occurs, and failure if the deployment fails.
- KITT generates a ServiceNow auto-closing change record when it deploys to a cluster with a profile of `prod`.
- The change record contains the commit SHA, the approver of the change, a link to the git repository, the version of the application, and the Concord log of the deployment.

kitt.yml

Multi App Deployments

- **Multiple services:** Multiple micro services in the same repository.
- **Multiple instances:** Single artifact, different runtime configurations.

kitt.yml

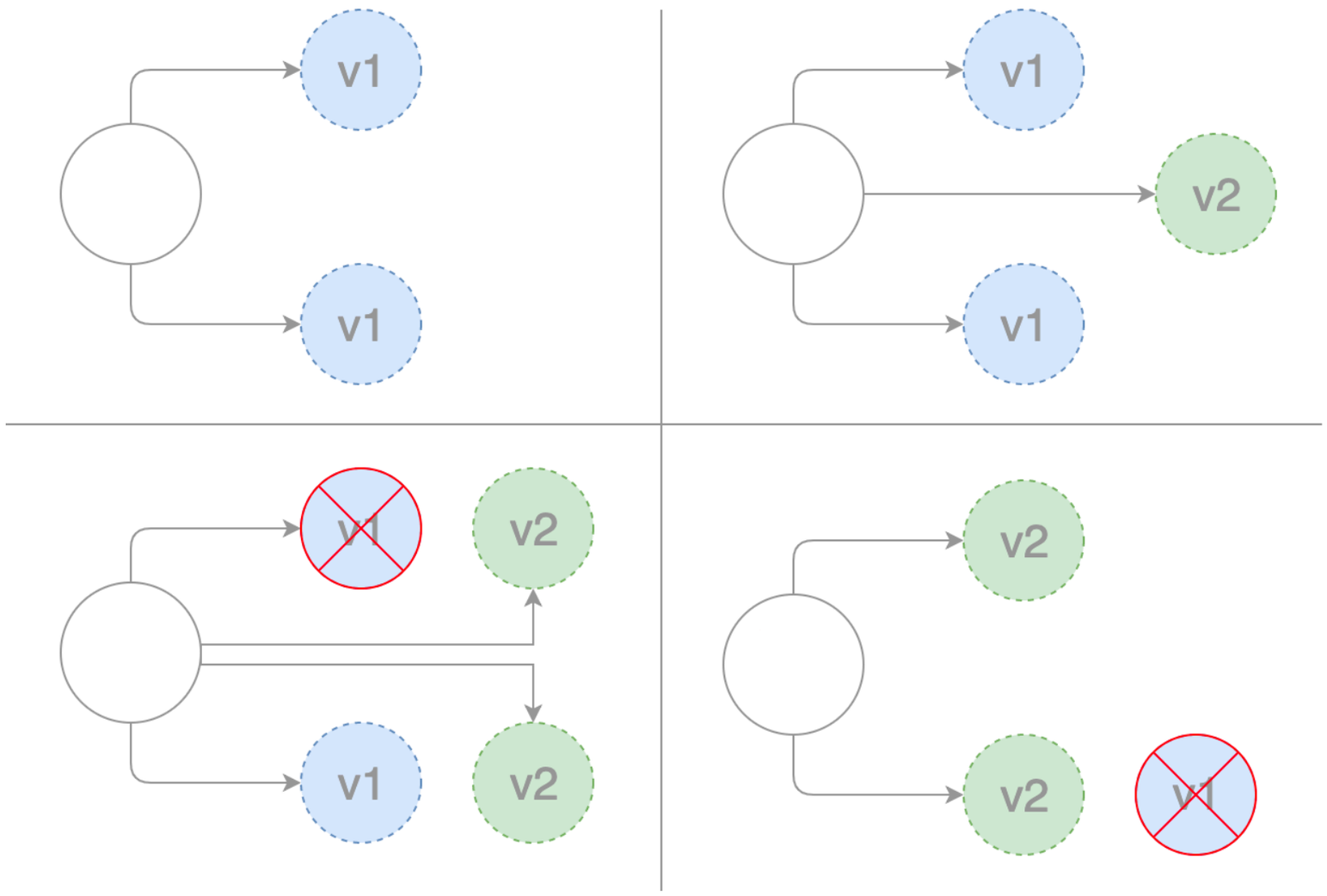
Release Strategies

Here's an overview of the most commonly used release strategies in the industry.

Release	Zero	Real Traffic	Targeted	Recommended For
Rolling Update /	✓	-	-	Internal services (safe, zero downtime, and simple)
Blue/Green	✓	-	-	Not recommended due to high cost and blast
Canary	✓	✓	-	For critical, customer-facing services (i.e.
**A/B Testing	✓	✓	✓	Long-running experiments (complimentary to

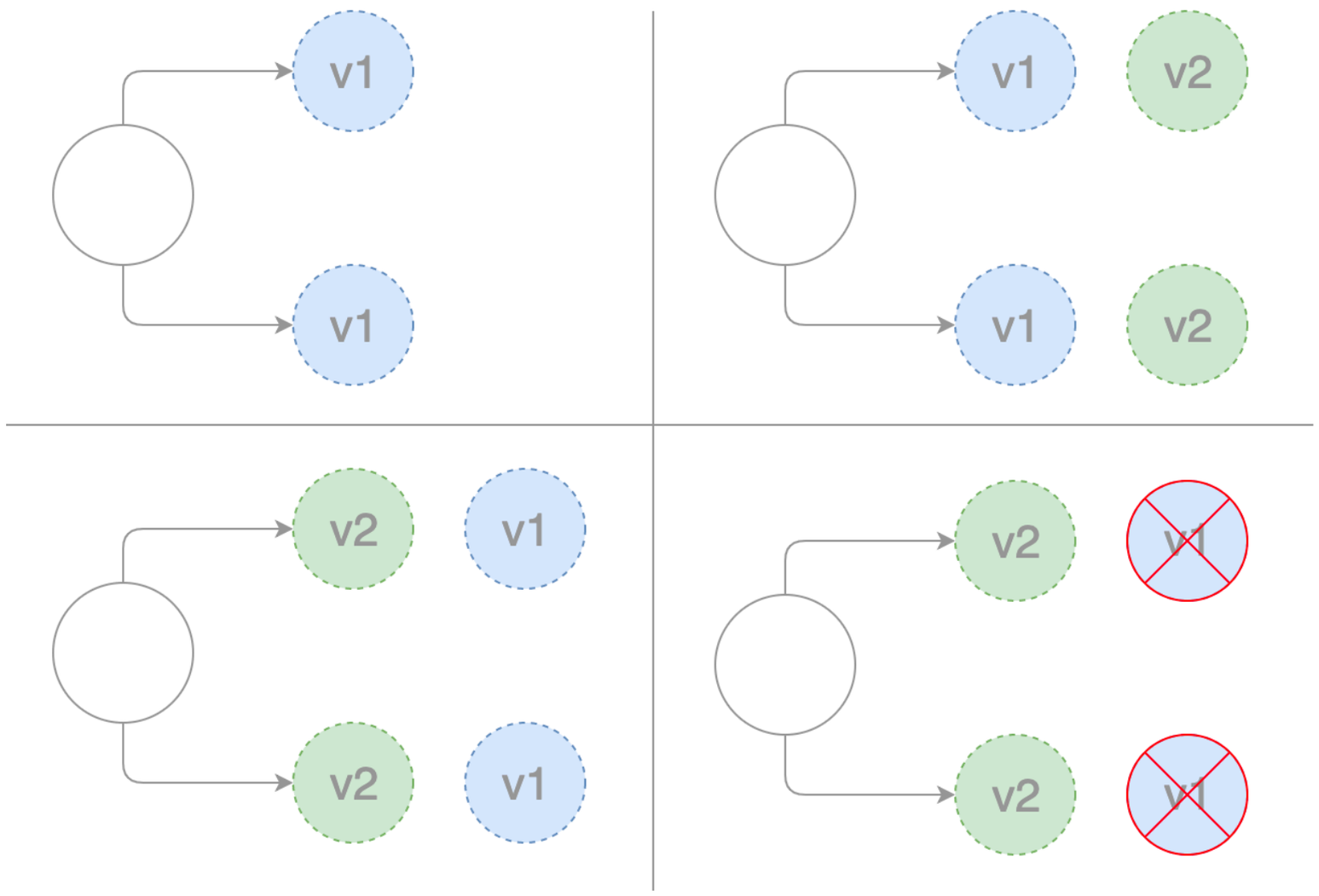
kitt.yml

Rolling Update V2 replaces V1



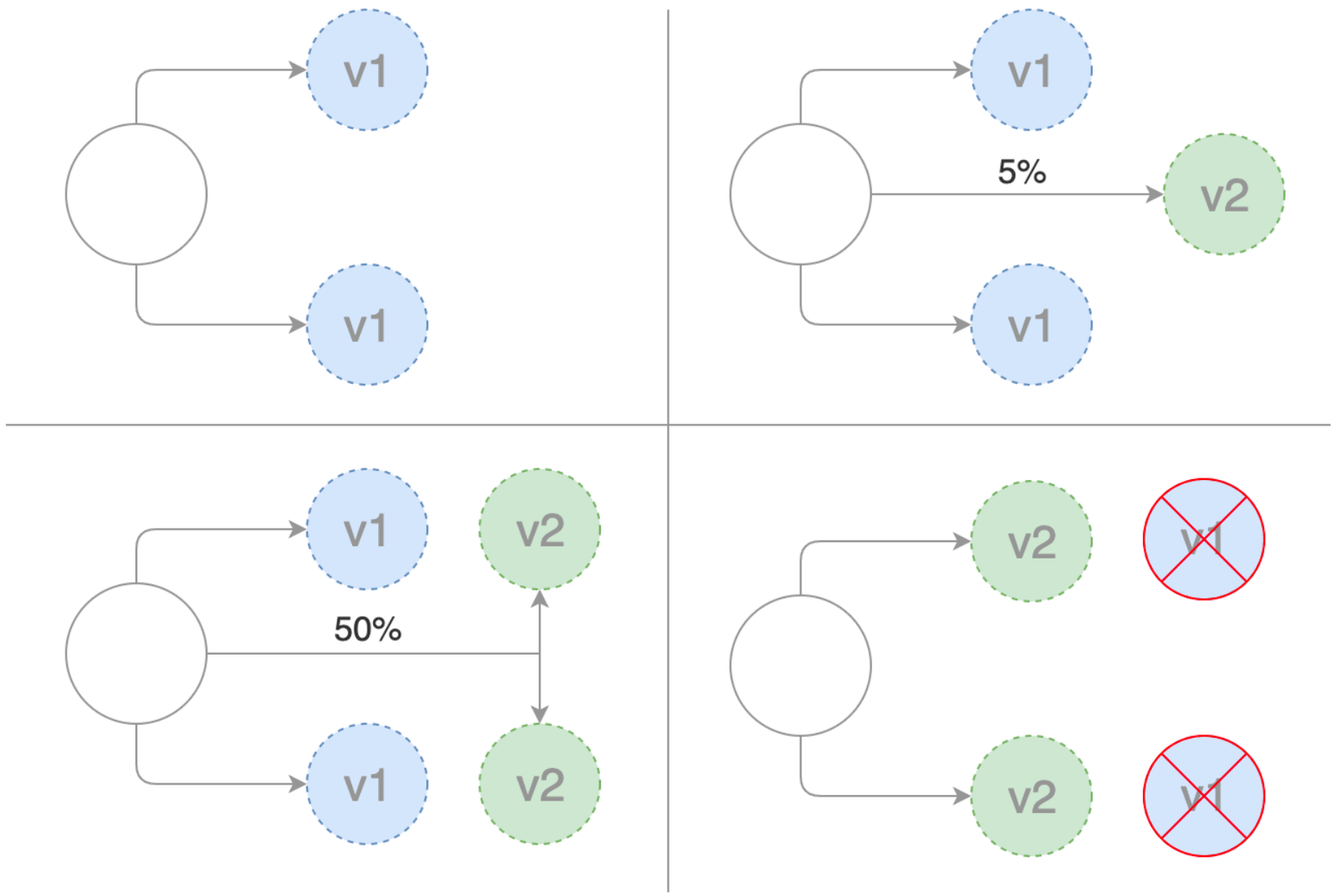
kitt.yml

Blue Green V2 replaces V1



kitt.yml

Canary V2 replaces V1



kitt.yml

A/B Testing

- Version V2 is released to a subset of users under specific conditions.
- A/B Testing is an experimentation technique that is designed for long-running experiments.
- A/B Testing is currently supported in WCNP by using a combination of KITT's multi-instance deployment and Expo.

kitt.yml

Recommendation

- Move to prod with rolling update
- Stable in prod? Try advanced release strategy

kitt.yml

Metrics with MMS

- MMS is based on Prometheus, uses M3DB as its centralized time series DB, and adopts Grafana for dashboarding.
- MMS gathers metadata about application performance when it runs on WCNP (such as CPU, memory and network usage)
- Provides options for enabling custom application metrics

kitt.yml

Metrics with MMS

MMS provides several built-in dashboards. The data comes from container metadata metrics:

- Apps
- Pods
- Istio Service Dashboard
- Custom Metrics

kitt.yml

Custom Metrics

```yaml
helm:
  values:
    global:
      metrics:
        enabled: false
        endpoints:
          - targetPort: 8080
            path: "/metrics"
        remoteWriteSampleLimit: 10
        whitelist:
          - metric_name_1
          - metric_name_2
```

kitt.yml

Custom Metrics

- `dev:` The prometheus will be created for you automatically if you enable custom metrics in kitt.yml.

- `stage|prod:` Provisioned namespace and enable collection of custom metrics from your namespace.

[Example Prometheus Namespace Url]

kitt.yml

Tracing and APM with Dynatrace

Applications deployed on WCNP using KITT are by default instrumented and monitored by Dynatrace. At the same time, automatically,

- A Dynatrace management zone with your namespace is created, for you to filter your services and any other Dynatrace monitored entities associated with your namespace, and
- A Spotlight alert subscription is create for your namespace to be delivered to the slack channel specified in your KITT.yml.

If desired, you can opt-out from the default Dynatrace instrumentation.

kitt.yml

Tracing and APM with Dynatrace

- Automatic anomaly detection and automated root cause analysis.
- Auto discovery all containers running on Kubernetes.
- Self-learning capabilities automatically discover an application environment's end-to-end flow without the need for configuration.
- All services, hosts, and dependencies are automatically visualized in an interactive topology map.
- Dynatrace's Analyze backtrace helps to find out who directly and indirectly calls a service.

kitt.yml

Alerts

- Alerts section provided at the root level will setup alert subscriptions in Spotlight for executing stage in your deployment.
- Stage-level, in which case stage-level attributes will override what is specified here.

```
alerts
  email:
    - noreply@walmart.com

  slackChannel:
    - alert-alert-channel
```

kitt.yml

Default Alerts

- Default alerts will trigger based on the health of the applications' pods
- failures to update during deployment.
- If your application:
  - seems unhealthy e.g. won't start
  - doesn't have enough resources
  - crashing often.
- issues related to gathering custom metrics.

etc.

kitt.yml

Custom Alerts

telemetry/mms-config gitops process.

```yaml
groups:
  - name: prod-service-name-wcnp-alert
    rules:
      - alert: <alert_name: http-req-5xx-rate>
        annotations:
          message: <custom_message>
          summary: <custom_summary>
          description: <custom_description such as: http://url_to_playbook>
        expr: <promql_expression_for_custom_metric>
        for: 5m
        labels:
          severity: <warning|critical>
          alert_team: <team_name>
          mms_slack_channel: <slack_channel_for_alerts>
          mms_xmatters_group: <xmatters_group_name>
```

kitt.yml

Alerts

Recommended approach:

1. Enable default alerts in KITT by routing them to Slack during development and testing.
2. Validate the alerts make sense and trigger at appropriate times. Choose the ones you require for going to production.
3. Build custom alerts using telemetry/mms-config, and configure routes for these to Slack and xMatters.

POD Profiling

- Measures the CPU and memory resource consumption in POD when under load.
- Need to know daily/peak traffic and response time SLA for your service to run POD profiler.
- Create performance test scenario for Automaton.

POD Profiling

- Provides recommendation to set the min/max resource values of all containers within a POD:

```
helm:
  values:
     min:
       cpu: 1
       memory: 512Mi
     max:
       cpu: 2
       memory: 2048Mi
```

POD Profiling

- Recommends required number of PODS for serving daily peak traffic and holiday peak traffic.
  - Autoscaling configuration
  - Namespace quotas

```
scaling:
  enabled: "true"
  max: 15
  min: 1
```

Summary

- Software Architecture
- DevOps Process

Summary

Software Architecture

- WCNP provides a Cloud Native runtime environment for Docker containers on Kubernetes. This implies that you are using a microservices architecture.
- At a minimum your application needs to be
  - Able to run in a Docker container within the available size limits.
  - Stateless

Summary

Development

- Dev environments. Tests with Docker
- Dev teams expected to learn to build, run, debug micro services in k8s with docker.
- WCNP Tools

Required Training

- Cloud Computing and Cloud Native Fundamentals Training
- Introduction to WCNP
- WCNP User Training: Level 2 - KITT

Summary

DevOps Process

KITT is an opinionated system that implements DevOps best practices:

- Trunk-based development
- Continuous Deployment
  git ops: commit -> dev -> stage -> prod
- Dev teams own and manage CICD pipeline (kitt)
- Dev team manages configuration via kitt (env) or CCM
- Dev teams maintain and support their environments.

QA

Thank you!