Automating SLI/SLO based build validation with Keptn

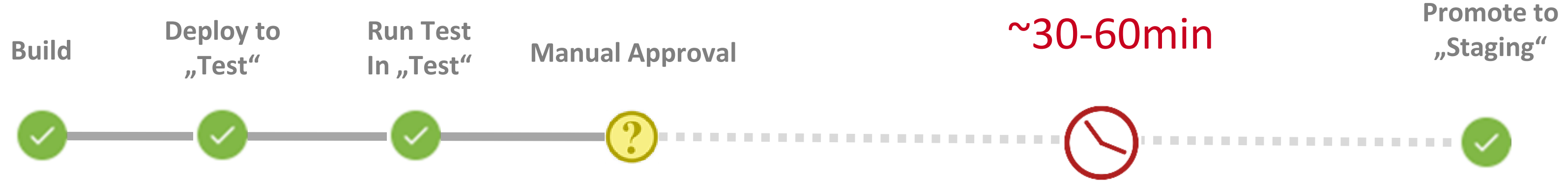
Cloud Native Bern Meetup
January 19th, 2021

Robin Wyss
Sales Engineer at Dynatrace

Web	http://keptn.sh/
Twitter	@keptnProject
GitHub	https://github.com/keptn/keptn
Tutorials	https://tutorials.keptn.sh
Slack	http://slack.keptn.sh

Lengthy manual approval

Jenkins GitLab



Build

Deploy to „Test“

Run Test In „Test“

Manual Approval

~30-60min

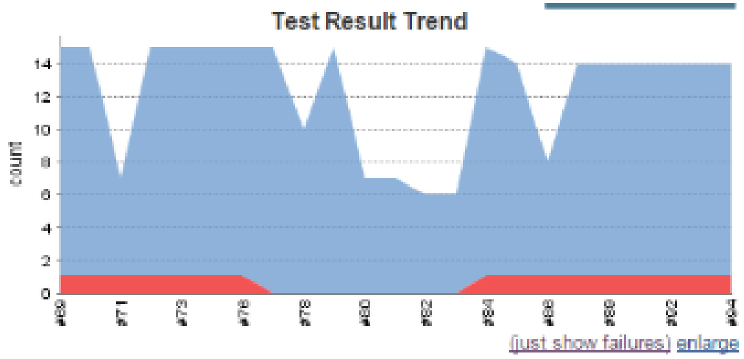
Promote to „Staging“

Is this regression impacting key business use cases ?

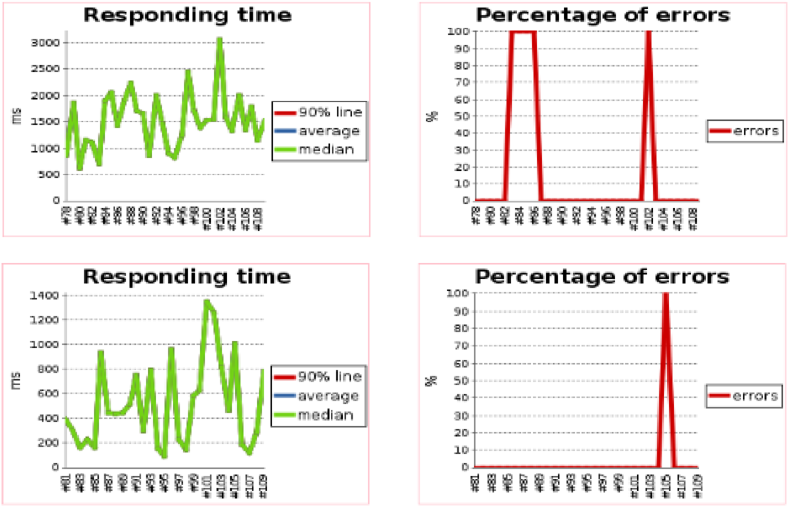
Which metrics are important and which build is therefore better ?

Which data comes from my test and is relevant for business transactions ?

Functional: Test Result Trend Not Enough

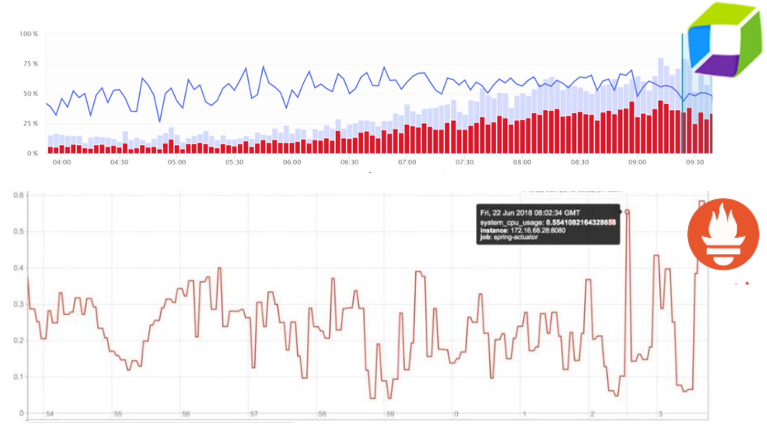


Performance: Manual Comparison Is Slow

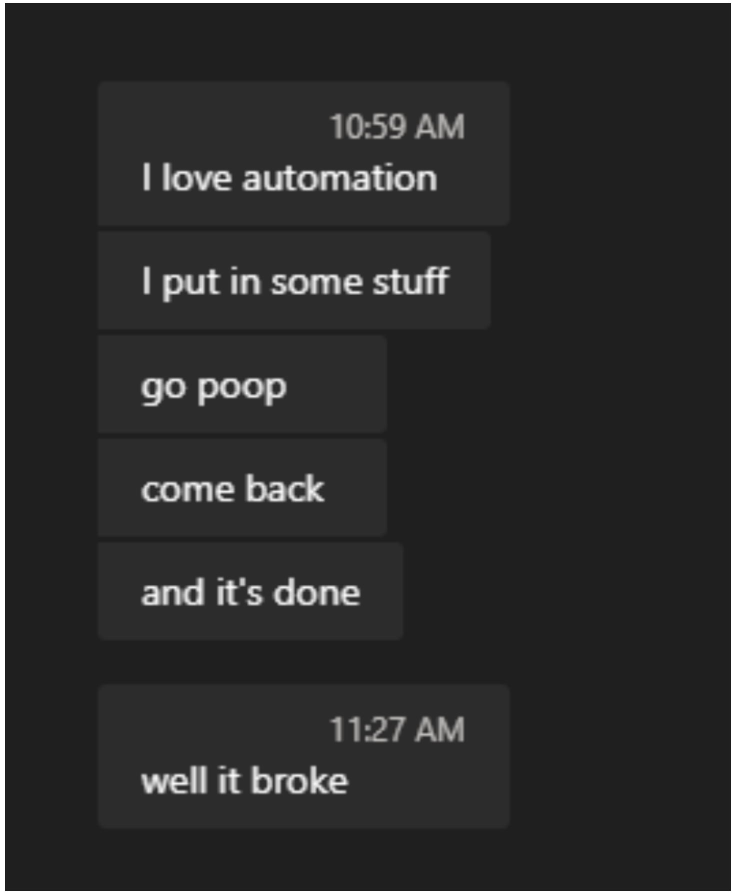


JMeter NeoLoad Gatling LOAD RUNNER

Monitoring: Too much unstructed data



Automation



10:59 AM

I love automation

I put in some stuff

go poop

come back

and it's done

11:27 AM

well it broke

Why Keptn?

2020 State of DevOps Report

63%

Building internal **self-service** delivery platforms

Success through Self-Service of

- Continuous Progressive Delivery
- Private & Cloud Infrastructure
- Development Environments
- Monitoring & Alerting
- Audit logging

Challenges due to lack of

- Time
- Standardization
- Technical skills

State of DevOps Report 2020: https://puppet.com/resources/report/2020-state-of-devops-report/

Why Keptn?

2020 State of DevOps Report

63%

Building internal **self-service** delivery platforms

Success through Self-Service of

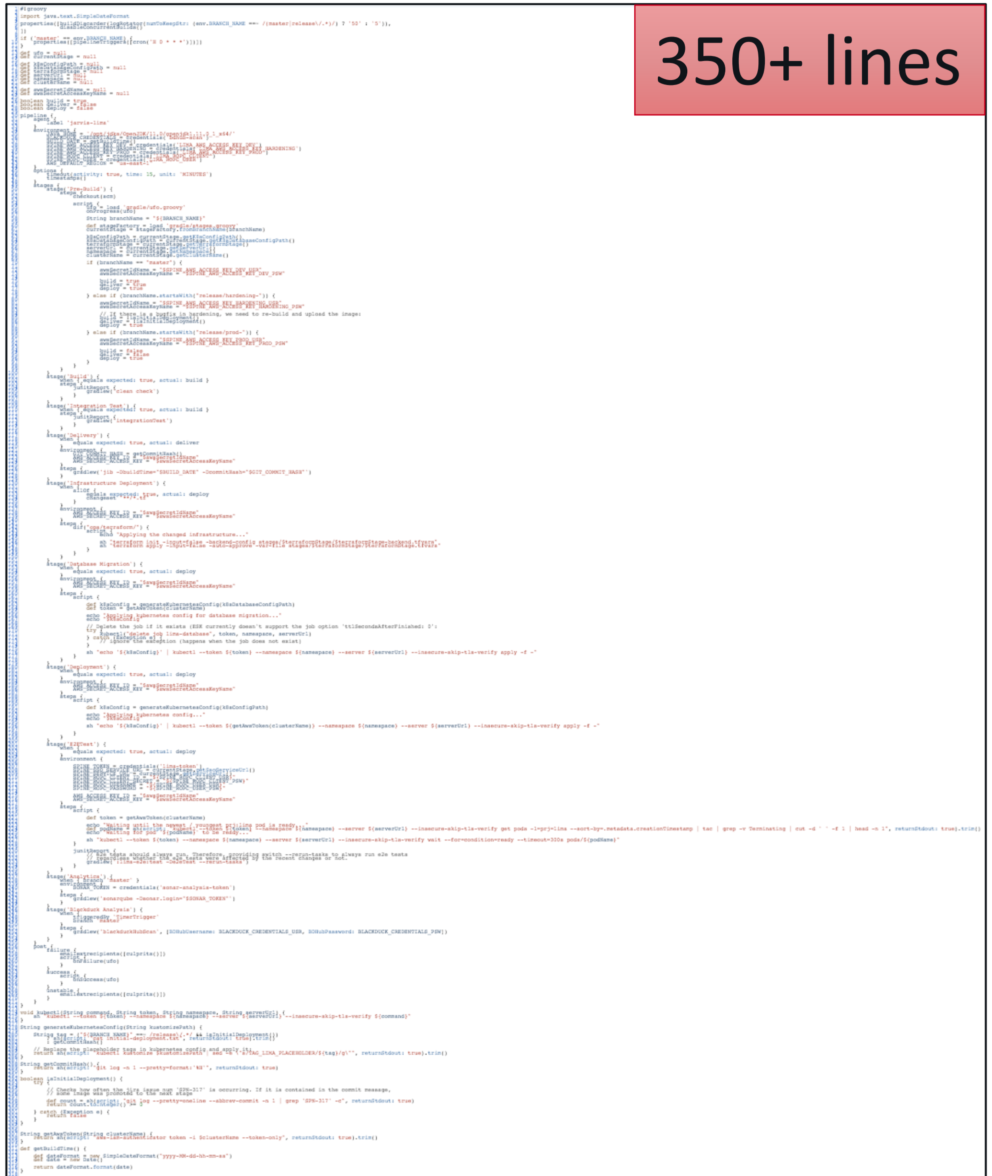
- Continuous Progressive Delivery
- Private & Cloud Infrastructure
- Development Environments
- Monitoring & Alerting
- Audit logging

keptn takes care of these challenges

- Time
- Standardization
- Technical skills

State of DevOps Report 2020: https://puppet.com/resources/report/2020-state-of-devops-report/

Delivery pipelines look like their monolithic source code counterparts



350+ lines

```groovy
#!groovy
import java.text.SimpleDateFormat
properties([buildDiscarder(logRotator(numToKeepStr: (env.BRANCH_NAME ==~ /(master|release\/.*)/) ? '50' : '5')),
            disableConcurrentBuilds()
])
if ('master' == env.BRANCH_NAME) {
    properties([pipelineTriggers([cron('H 0 * * *')])])
}
def ufo = null
def currentStage = null
def k8sConfigPath = null
def k8sDatabaseConfigPath = null
def terraformStage = null
def serverUrl = null
def namespace = null
def clusterName = null
def awsSecretIdName = null
def awsSecretAccessKeyName = null
boolean build = true
boolean deliver = false
boolean deploy = false
pipeline {
    agent {
        label 'jarvis-lima'
    }
    environment {
        JAVA_HOME = '/opt/jdks/OpenJDK/11.0/openjdk1.11.0_1_x64/'
        BLACKDUCK_CREDENTIALS = credentials('bdhub-scan')
        BUILD_DATE = getBuildTime()
        SPINE_AWS_ACCESS_KEY_DEV = credentials('LIMA_AWS_ACCESS_KEY_DEV')
        SPINE_AWS_ACCESS_KEY_HARDENING = credentials('LIMA_AWS_ACCESS_KEY_HARDENING')
        SPINE_AWS_ACCESS_KEY_PROD = credentials('LIMA_AWS_ACCESS_KEY_PROD')
        SPINE_SOPC_CLIENT = credentials('LIMA_SOPC_CLIENT')
        SPINE_SOPC_USER = credentials('LIMA_SOPC_USER')
        AWS_DEFAULT_REGION = 'us-east-1'
    }
    options {
        timeout(activity: true, time: 15, unit: 'MINUTES')
        timestamps()
    }
    stages {
        stage('Pre-Build') {
            steps {
                checkout(scm)

                script {
                    ufo = load 'gradle/ufo.groovy'
                    onProgress(ufo)

                    String branchName = "${BRANCH_NAME}"

                    def stageFactory = load 'gradle/stages.groovy'
                    currentStage = stageFactory.fromBranchName(branchName)

                    k8sConfigPath = currentStage.getK8sConfigPath()
                    k8sDatabaseConfigPath = currentStage.getK8sDatabaseConfigPath()
                    terraformStage = currentStage.getTerraformStage()
                    serverUrl = currentStage.getServerUrl()
                    namespace = currentStage.getNamespace()
                    clusterName = currentStage.getClusterName()

                    if (branchName == "master") {

                        awsSecretIdName = "$SPINE_AWS_ACCESS_KEY_DEV_USR"
                        awsSecretAccessKeyName = "$SPINE_AWS_ACCESS_KEY_DEV_PSW"

                        build = true
                        deliver = true
                        deploy = true

                    } else if (branchName.startsWith("release/hardening-")) {

                        awsSecretIdName = "$SPINE_AWS_ACCESS_KEY_HARDENING_USR"
                        awsSecretAccessKeyName = "$SPINE_AWS_ACCESS_KEY_HARDENING_PSW"

                        // If there is a bugfix in hardening, we need to re-build and upload the image:
                        build = !isInitialDeployment()
                        deliver = !isInitialDeployment()
                        deploy = true

                    } else if (branchName.startsWith("release/prod-")) {

                        awsSecretIdName = "$SPINE_AWS_ACCESS_KEY_PROD_USR"
                        awsSecretAccessKeyName = "$SPINE_AWS_ACCESS_KEY_PROD_PSW"

                        build = false
                        deliver = false
                        deploy = true
                    }
                }
            }
        }
        stage('Build') {
            when { equals expected: true, actual: build }
            steps {
                junitReport {
                    gradlew('clean check')
                }
            }
        }
        stage('Integration Test') {
            when { equals expected: true, actual: build }
            steps {
                junitReport {
                    gradlew('integrationTest')
                }
            }
        }
        stage('Delivery') {
            when {
                equals expected: true, actual: deliver
            }
            environment {
                GIT_COMMIT_HASH = getCommitHash()
                AWS_ACCESS_KEY_ID = "$awsSecretIdName"
                AWS_SECRET_ACCESS_KEY = "$awsSecretAccessKeyName"
            }
            steps {
                gradlew('jib -DbuildTime="$BUILD_DATE" -DcommitHash="$GIT_COMMIT_HASH"')
            }
        }
        stage('Infrastructure Deployment') {
            when {
                allOf {
                    equals expected: true, actual: deploy
                    changeset "**/*.tf"
                }
            }
            environment {
                AWS_ACCESS_KEY_ID = "$awsSecretIdName"
                AWS_SECRET_ACCESS_KEY = "$awsSecretAccessKeyName"
            }
            steps {
                dir("ops/terraform/") {
                    script {
                        echo "Applying the changed infrastructure..."

                        sh "terraform init -input=false -backend-config stages/$terraformStage/$terraformStage-backend.tfvars"
                        sh "terraform apply -input=false -auto-approve -var-file stages/$terraformStage/$terraformStage.tfvars"
                    }
                }
            }
        }
        stage('Database Migration') {
            when {
                equals expected: true, actual: deploy
            }
            environment {
                AWS_ACCESS_KEY_ID = "$awsSecretIdName"
                AWS_SECRET_ACCESS_KEY = "$awsSecretAccessKeyName"
            }
            steps {
                script {
                    def k8sConfig = generateKubernetesConfig(k8sDatabaseConfigPath)
                    def token = getAwsToken(clusterName)

                    echo "Applying kubernetes config for database migration..."
                    echo "$k8sConfig"

                    // Delete the job if it exists (ESK currently doesn't support the job option 'ttlSecondsAfterFinished: 0':
                    try {
                        kubectl("delete job lima-database", token, namespace, serverUrl)
                    } catch (Exception e) {
                        // ignore the exception (happens when the job does not exist)
                    }

                    sh "echo '${k8sConfig}' | kubectl --token ${token} --namespace ${namespace} --server ${serverUrl} --insecure-skip-tls-verify apply -f -"
                }
            }
        }
        stage('Deployment') {
            when {
                equals expected: true, actual: deploy
            }
            environment {
                AWS_ACCESS_KEY_ID = "$awsSecretIdName"
                AWS_SECRET_ACCESS_KEY = "$awsSecretAccessKeyName"
            }
            steps {
                script {
                    def k8sConfig = generateKubernetesConfig(k8sConfigPath)

                    echo "Applying kubernetes config..."
                    echo "$k8sConfig"

                    sh "echo '${k8sConfig}' | kubectl --token ${getAwsToken(clusterName)} --namespace ${namespace} --server ${serverUrl} --insecure-skip-tls-verify apply -f -"

                }
            }
        }
        stage('E2ETest') {
            when {
                equals expected: true, actual: deploy
            }
            environment {
                SPINE_TOKEN = credentials('lima-token')
                SPINE_SSO_SERVICE_URL = currentStage.getSsoServiceUrl()
                SPINE_SERVICE_URL = currentStage.getServiceUrl()
                SPINE_SOPC_CLIENT_ID = "${SPINE_SOPC_CLIENT_USR}"
                SPINE_SOPC_CLIENT_SECRET = "${SPINE_SOPC_CLIENT_PSW}"
                SPINE_SOPC_USERNAME = "${SPINE_SOPC_USER_USR}"
                SPINE_SOPC_PASSWORD = "${SPINE_SOPC_USER_PSW}"

                AWS_ACCESS_KEY_ID = "$awsSecretIdName"
                AWS_SECRET_ACCESS_KEY = "$awsSecretAccessKeyName"
            }
            steps {
                script {
                    def token = getAwsToken(clusterName)

                    echo "Waiting until the newest / youngest prj:lima pod is ready..."
                    def podName = sh(script: "kubectl --token ${token} --namespace ${namespace} --server ${serverUrl} --insecure-skip-tls-verify get pods -l=prj=lima --sort-by=.metadata.creationTimestamp | tac | grep -v Terminating | cut -d ' ' -f 1 | head -n 1", returnStdout: true).trim()
                    echo "Waiting for pod '${podName}' to be ready..."

                    sh "kubectl --token ${token} --namespace ${namespace} --server ${serverUrl} --insecure-skip-tls-verify wait --for=condition=ready --timeout=300s pods/${podName}"
                }

                junitReport {
                    // e2e tests should always run. Therefore, providing switch --rerun-tasks to always run e2e tests
                    // regardless whether the e2e tests were affected by the recent changes or not.
                    gradlew(':lima-e2e:test -De2eTest --rerun-tasks')
                }
            }
        }
        stage('Analytics') {
            when { branch 'master' }
            environment {
                SONAR_TOKEN = credentials('sonar-analysis-token')
            }
            steps {
                gradlew('sonarqube -Dsonar.login="$SONAR_TOKEN"')
            }
        }
        stage('Blackduck Analysis') {
            when {
                triggeredBy 'TimerTrigger'
                branch 'master'
            }
            steps {
                gradlew('blackduckHubScan', [BDHubUsername: BLACKDUCK_CREDENTIALS_USR, BDHubPassword: BLACKDUCK_CREDENTIALS_PSW])
            }
        }
    }
    post {
        failure {
            emailextrecipients([culprits()])
            script {
                onFailure(ufo)
            }
        }
        success {
            script {
                onSuccess(ufo)
            }
        }
        unstable {
            emailextrecipients([culprits()])
        }
    }
}
void kubectl(String command, String token, String namespace, String serverUrl) {
    sh "kubectl --token ${token} --namespace ${namespace} --server ${serverUrl} --insecure-skip-tls-verify ${command}"
}
String generateKubernetesConfig(String kustomizePath) {

    String tag = ("${BRANCH_NAME}" ==~ /release\/.*/ && isInitialDeployment())
            ? sh(script: "cat initial-deployment.txt", returnStdout: true).trim()
            : getCommitHash()

    // Replace the placeholder tags in kubernetes config and apply it:
    return sh(script: "kubectl kustomize $kustomizePath | sed -e \"s/TAG_LIMA_PLACEHOLDER/${tag}/g\"", returnStdout: true).trim()
}
String getCommitHash() {
    return sh(script: "git log -n 1 --pretty=format:'%H'", returnStdout: true)
}
boolean isInitialDeployment() {
    try {
        // Checks how often the jira issue num 'SPN-317' is occurring. If it is contained in the commit message,
        // some image was promoted to the next stage
        def count = sh(script: "git log --pretty=oneline --abbrev-commit -n 1 | grep 'SPN-317' -c", returnStdout: true)
        return count.toInteger() >= 0
    } catch (Exception e) {
        return false
    }
}

String getAwsToken(String clusterName) {
    return sh(script: "aws-iam-authenticator token -i $clusterName --token-only", returnStdout: true).trim()
}
def getBuildTime() {
    def dateFormat = new SimpleDateFormat("yyyy-MM-dd-hh-mm-ss")
    def date = new Date()
    return dateFormat.format(date)
}
```

Mixed **information** about

- **Process** (build, deploy, test, evaluate, …)
- **Target platform** (k8s, …)
- **Environments** (dev, hardening, …)
- **Tools** (Terraform, Helm, hey, …)

No clear **separation** of **concerns**

- **Developers**
  - Define which artifact to use
  - Want fast feedback on their code
- **DevOps Engineers**
  - Define which tools to use
  - Ensure tools are properly configured
- **Site Reliability Engineers**
  - Define delivery processes
  - Define operations workflows

And we get a lot of copies that make it harder to maintain or fix issues

1 Service = 1 Pipeline

```
pipeline {
  stages {
    stage('Deploy to dev namespace') {
      steps {
        container('helm') {
        }
      }
    }
    stage('Run tests') {
      steps {
        container('hey') {
        }
      }
    }
    stage('Evaluate performance') {
      steps {
        container('curl') {
        }
      }
    }
    if (evaluation.passed) {
      stage('Deploy to staging') {
        steps {
          container('helm') {
          }
        }
      }
    }
  }
}
```

1 Project = x Pipelines

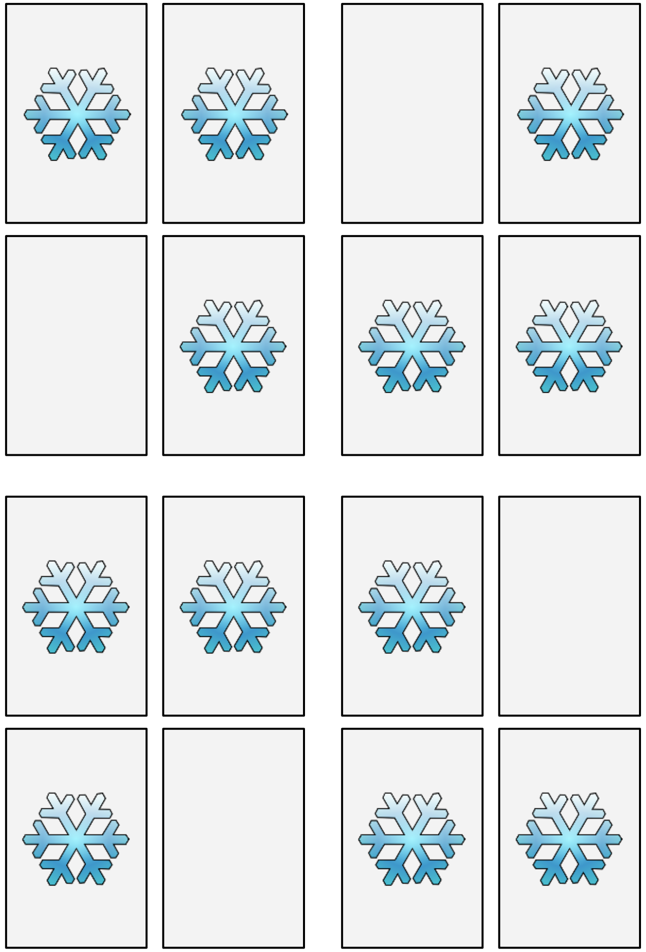
```
pipeline {
  stages {
    stage('Deploy to dev namespace') {
      steps {
        container('helm') {
        }
      }
    }
    stage('Run tests') {
      steps {
        container('jmeter') {
        }
      }
    }
    stage('Evaluate performance') {
      steps {
        container('curl') {
        }
      }
    }
    if (evaluation.passed) {
      stage('Deploy to staging') {
        steps {
          container('helm') {
          }
        }
      }
    }
  }
}
```

```
pipeline {
  stages {
    stage('Deploy to dev namespace') {
      steps {
        container('kustomize') {
        }
      }
    }
    stage('Run tests') {
      steps {
        container('jmeter') {
        }
      }
    }
    stage('Evaluate performance') {
      steps {
        container('curl') {
        }
      }
    }
    if (evaluation.passed) {
      stage('Deploy to staging') {
        steps {
          container('helm') {
          }
        }
      }
    }
  }
}
```

```
pipeline {
  stages {
    stage('Deploy to dev namespace') {
      steps {
        container('helm') {
        }
      }
    }
    stage('Run tests') {
      steps {
        container('jmeter') {
        }
      }
    }
    stage('Evaluate performance') {
      steps {
        container('curl') {
        }
      }
    }
    if (evaluation.passed) {
      stage('Deploy to staging') {
        steps {
          container('helm') {
          }
        }
      }
    }
  }
}
```

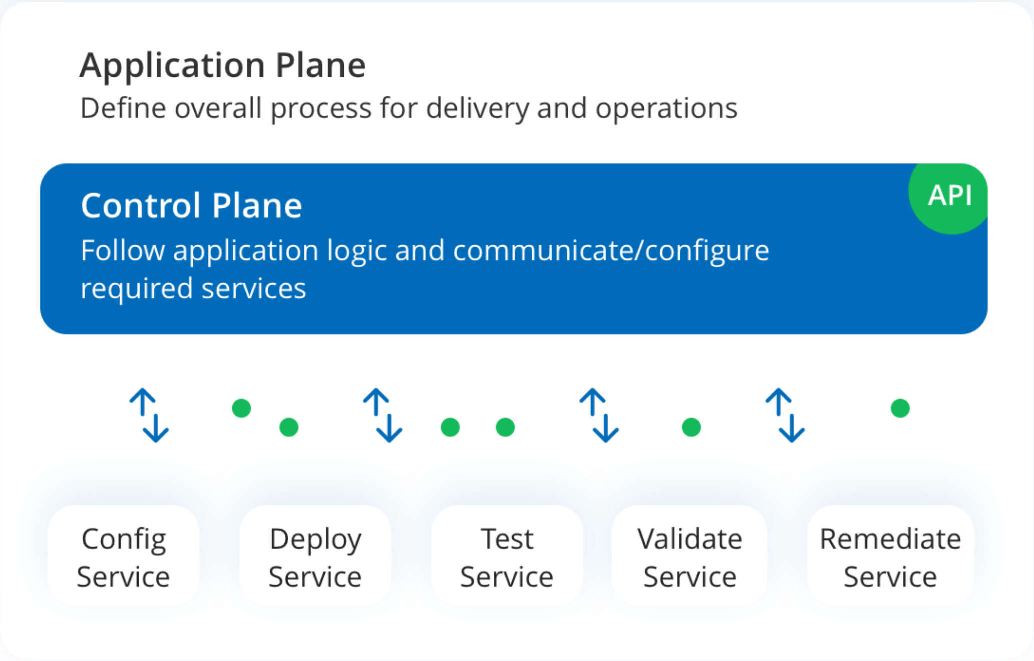
```
pipeline {
  stages {
    stage('Deploy to dev namespace') {
      steps {
        container('helm') {
        }
      }
    }
    stage('Run tests') {
      steps {
        container('selenium') {
        }
      }
    }

    if (evaluation.passed) {
      stage('Deploy to staging') {
        steps {
          container('helm') {
          }
        }
      }
    }
  }
}
```

n Teams = n*x Pipelines



Keptn in a nutshell

Keptn is an event-based **control plane** for **continuous delivery** and **automated operations** for cloud-native applications.



Application Plane
Define overall process for delivery and operations
Control Plane
Follow application logic and communicate/configure required services
API
Config Service
Deploy Service
Test Service
Validate Service
Remediate Service

CLOUD NATIVE
COMPUTING FOUNDATION

Keptn: Data-Driven Delivery & Operations Automation

pick your **use case**

- SLO-Quality Gates
- Progressive Delivery
- SRE Automation
- Auto-Remediation

You
(Dev/Ops/SRE)

bring your **configuration**

- shipyard
- SLI/SLO
- workload
- runbook

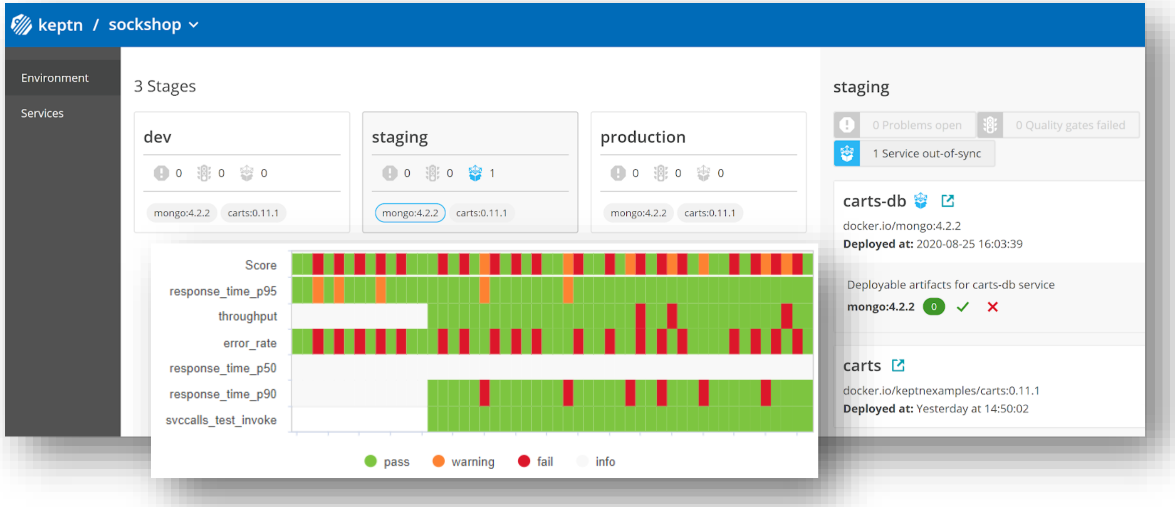
connect your **tools**

HELM, argo, JMeter …

keptn

automates configuration and provides **self-service** for

Monitoring | Delivery | Reliability | Remediation



keptn / sockshop

Environment
Services

3 Stages

dev: 0 0 0 — mongo:4.2.2 carts:0.11.1

staging: 0 0 1 — mongo:4.2.2 carts:0.11.1

production: 0 0 0 — mongo:4.2.2 carts:0.11.1

staging

0 Problems open
0 Quality gates failed
1 Service out-of-sync

carts-db
docker.io/mongo:4.2.2
Deployed at: 2020-08-25 16:03:39

Deployable artifacts for carts-db service
mongo:4.2.2 0 ✓ ✗

carts
docker.io/keptnexamples/carts:0.11.1
Deployed at: Yesterday at 14:50:02

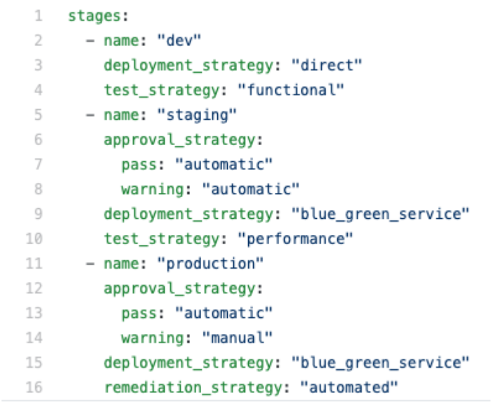
through **event-driven** process **orchestration** based on

Declaration | GitOps | SLOs | Standards

Automating SLO-driven Multi-stage Delivery

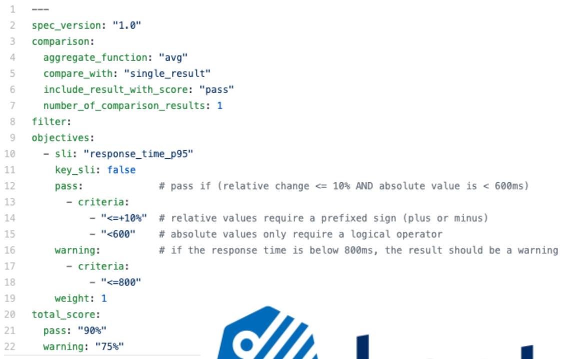
Automating SLO-driven Multi-stage Delivery

shipyard.yaml



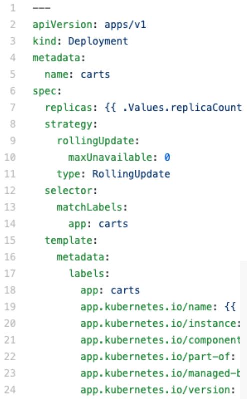
```yaml
stages:
  - name: "dev"
    deployment_strategy: "direct"
    test_strategy: "functional"
  - name: "staging"
    approval_strategy:
      pass: "automatic"
      warning: "automatic"
    deployment_strategy: "blue_green_service"
    test_strategy: "performance"
  - name: "production"
    approval_strategy:
      pass: "automatic"
      warning: "manual"
    deployment_strategy: "blue_green_service"
    remediation_strategy: "automated"
```

slo.yaml



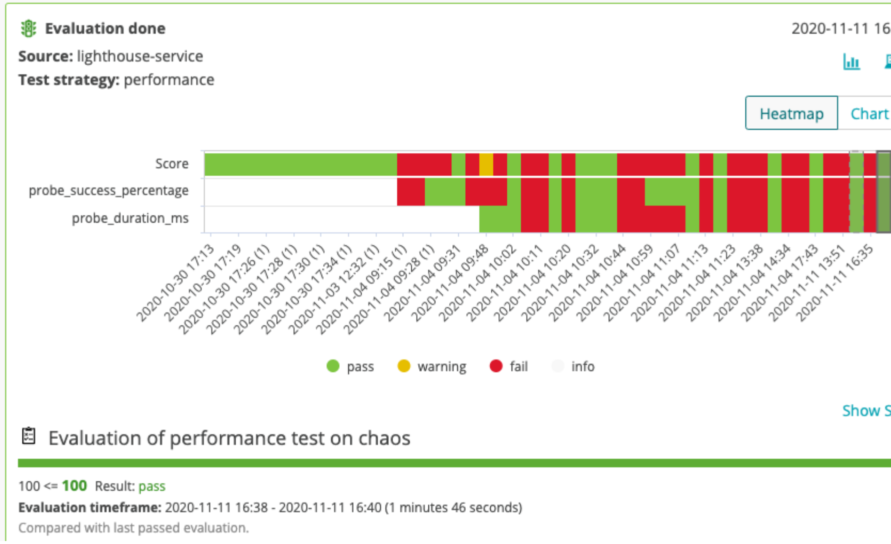
```yaml
---
spec_version: "1.0"
comparison:
  aggregate_function: "avg"
  compare_with: "single_result"
  include_result_with_score: "pass"
  number_of_comparison_results: 1
filter:
objectives:
  - sli: "response_time_p95"
    key_sli: false
    pass:             # pass if (relative change <= 10% AND absolute value is < 600ms)
      - criteria:
          - "<=+10%"  # relative values require a prefixed sign (plus or minus)
          - "<600"    # absolute values only require a logical operator
    warning:          # if the response time is below 800ms, the result should be a warning
      - criteria:
          - "<=800"
    weight: 1
total_score:
  pass: "90%"
  warning: "75%"
```

Helm chart



```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: carts
spec:
  replicas: {{ .Values.replicaCount }}
  strategy:
    rollingUpdate:
      maxUnavailable: 0
    type: RollingUpdate
  selector:
    matchLabels:
      app: carts
  template:
    metadata:
      labels:
        app: carts
        app.kubernetes.io/name: {{
        app.kubernetes.io/instance:
        app.kubernetes.io/component
        app.kubernetes.io/part-of:
        app.kubernetes.io/managed-b
        app.kubernetes.io/version:
```

keptn



Evaluation done

2020-11-11 16

Source: lighthouse-service

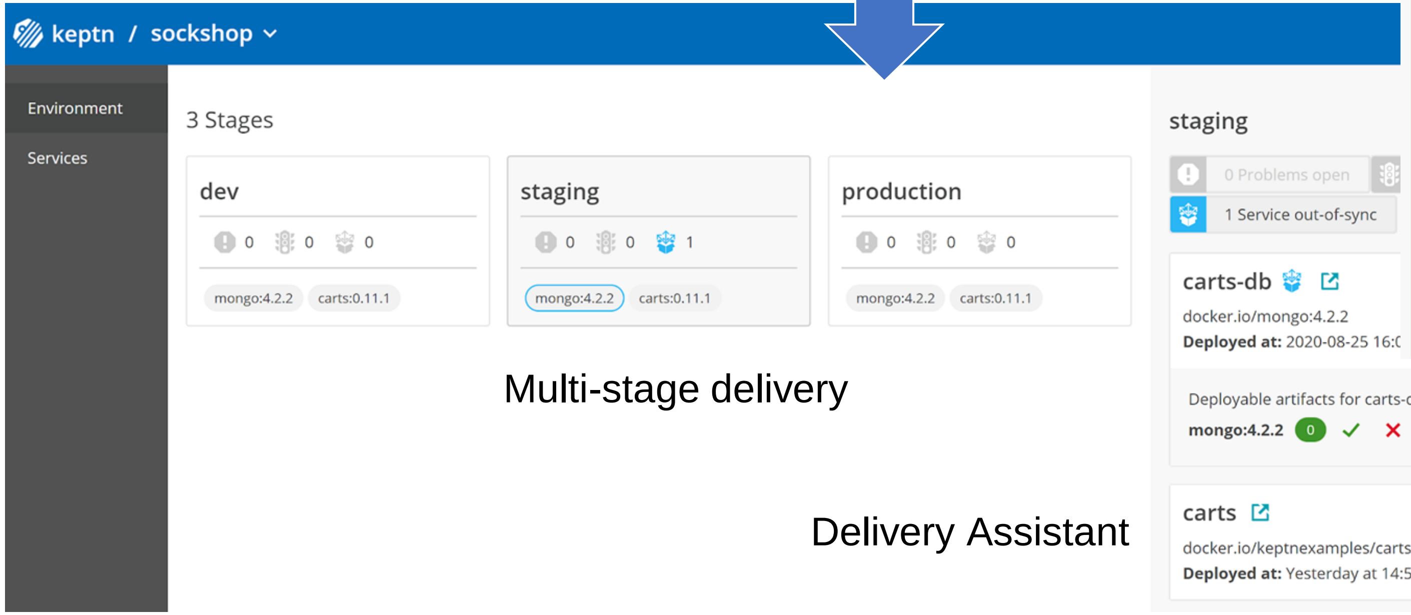
Test strategy: performance

Evaluation of performance test on chaos

100 <= **100** Result: pass

Evaluation timeframe: 2020-11-11 16:38 - 2020-11-11 16:40 (1 minutes 46 seconds)

Compared with last passed evaluation.



3 Stages

dev	staging	production
0 0 0	0 0 1	0 0 0
mongo:4.2.2 carts:0.11.1	mongo:4.2.2 carts:0.11.1	mongo:4.2.2 carts:0.11.1

staging

0 Problems open

1 Service out-of-sync

carts-db

docker.io/mongo:4.2.2

Deployed at: 2020-08-25 16:(

Deployable artifacts for carts-db service

mongo:4.2.2 0 ✓ ✗

carts

docker.io/keptnexamples/carts:0.11.1

Deployed at: Yesterday at 14:50:02

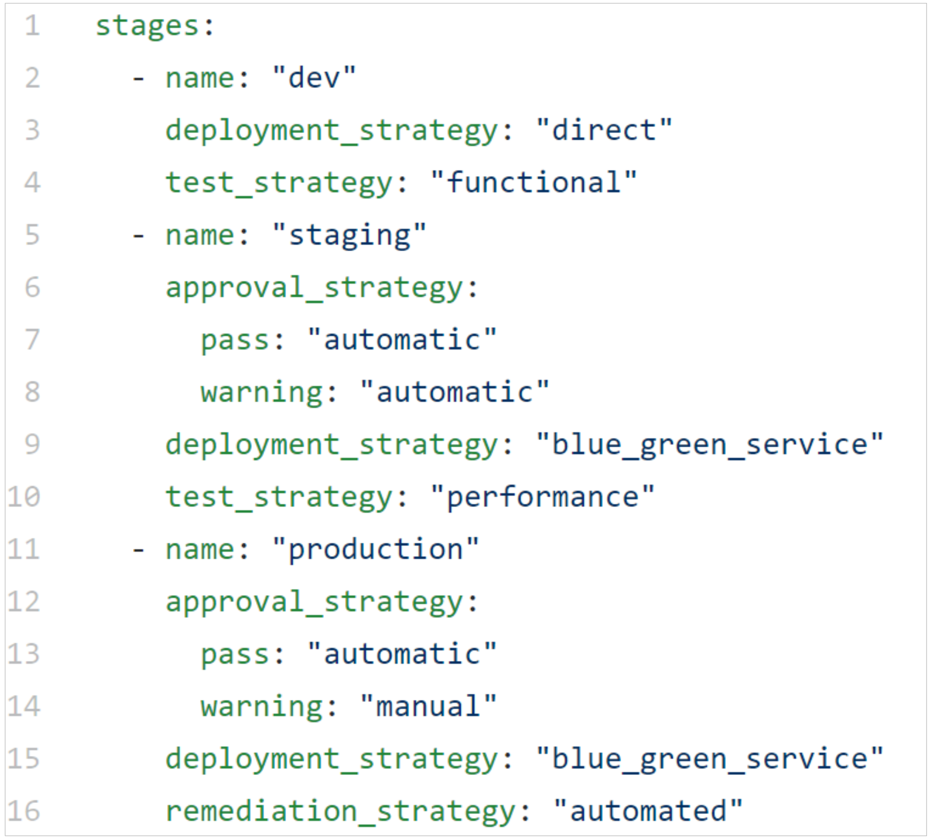
Multi-stage delivery

Integrated Quality Gates

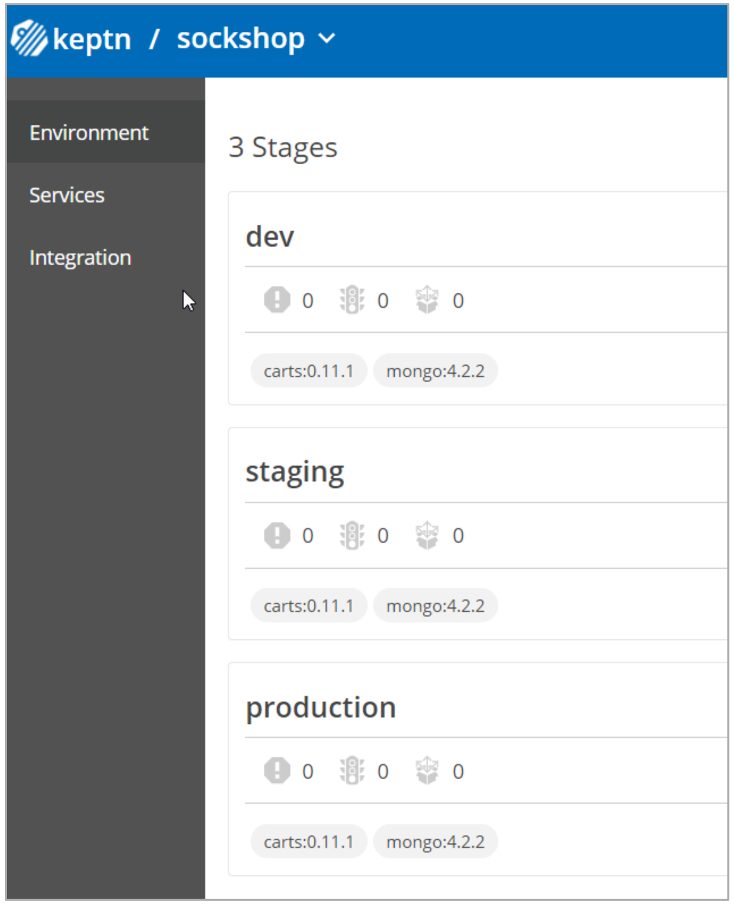
Delivery Assistant

Create project

```
$ keptn create project sockshop --shipyard=./shipyard.yaml [--git-user=...]
```



```yaml
stages:
  - name: "dev"
    deployment_strategy: "direct"
    test_strategy: "functional"
  - name: "staging"
    approval_strategy:
      pass: "automatic"
      warning: "automatic"
    deployment_strategy: "blue_green_service"
    test_strategy: "performance"
  - name: "production"
    approval_strategy:
      pass: "automatic"
      warning: "manual"
    deployment_strategy: "blue_green_service"
    remediation_strategy: "automated"
```



3 Stages

dev

0 0 0

carts:0.11.1 mongo:4.2.2

staging

0 0 0

carts:0.11.1 mongo:4.2.2

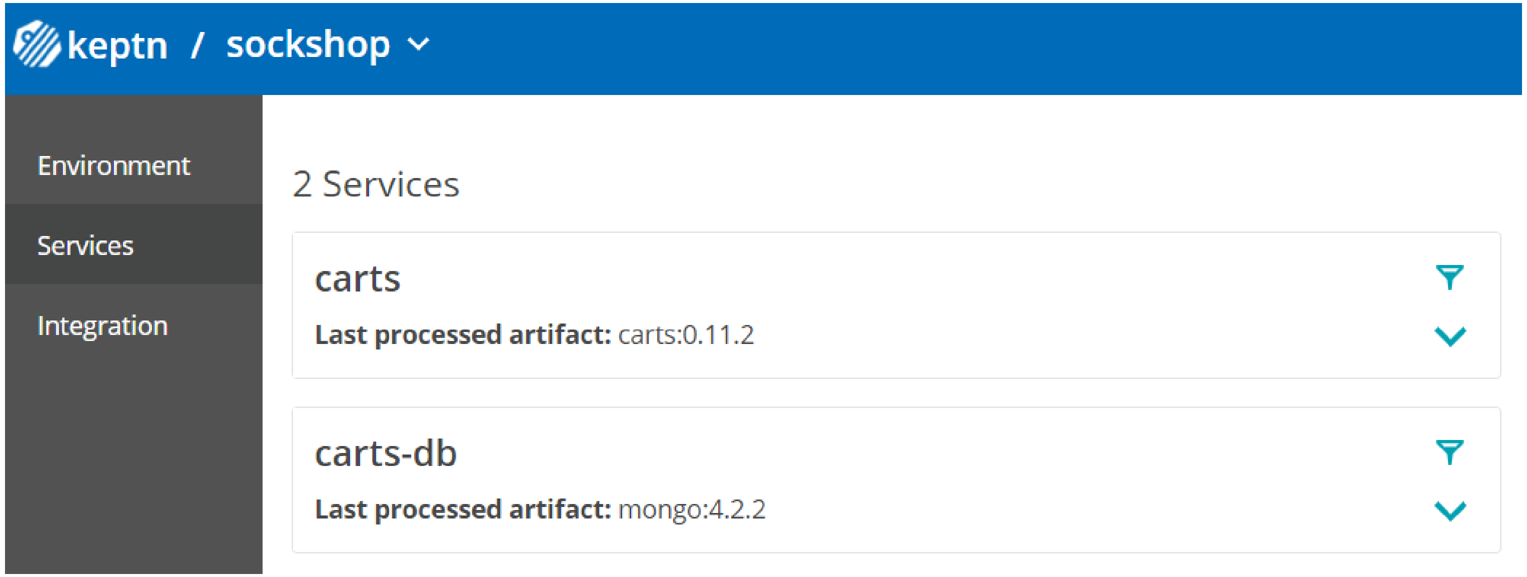
production

0 0 0

carts:0.11.1 mongo:4.2.2

Add services

```
$ keptn onboard service carts --project=sockshop --chart=./carts
```



2 Services

carts

Last processed artifact: carts:0.11.2

carts-db

Last processed artifact: mongo:4.2.2

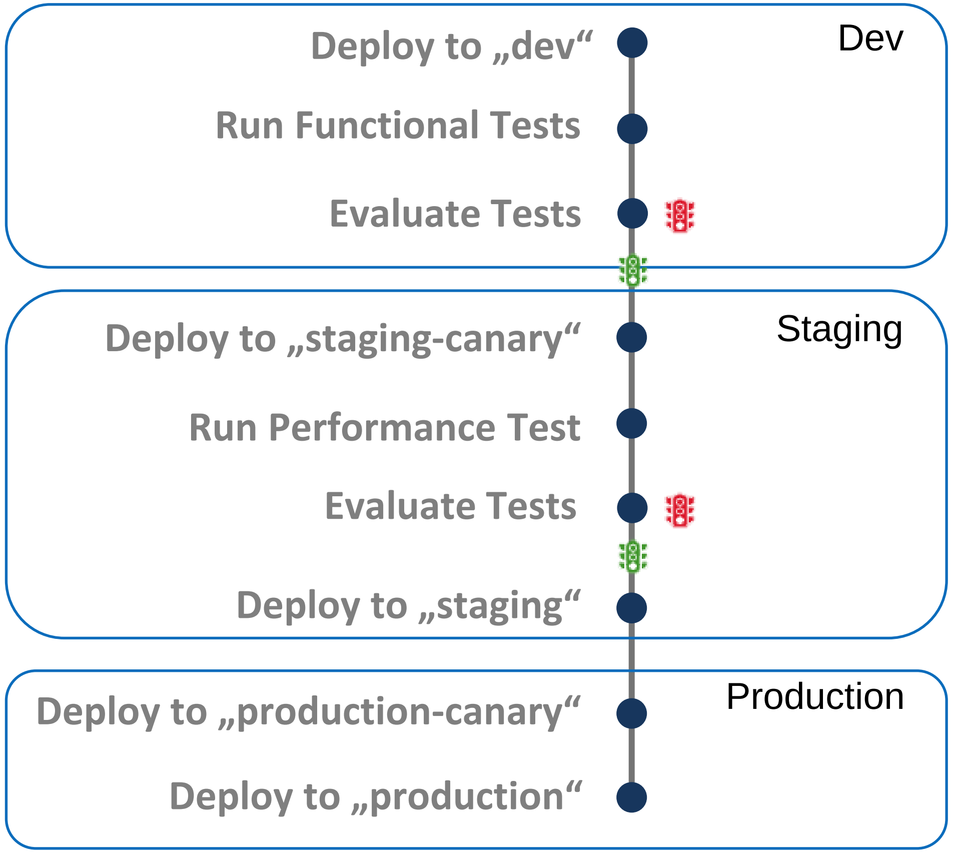
Add tests (jmeter)

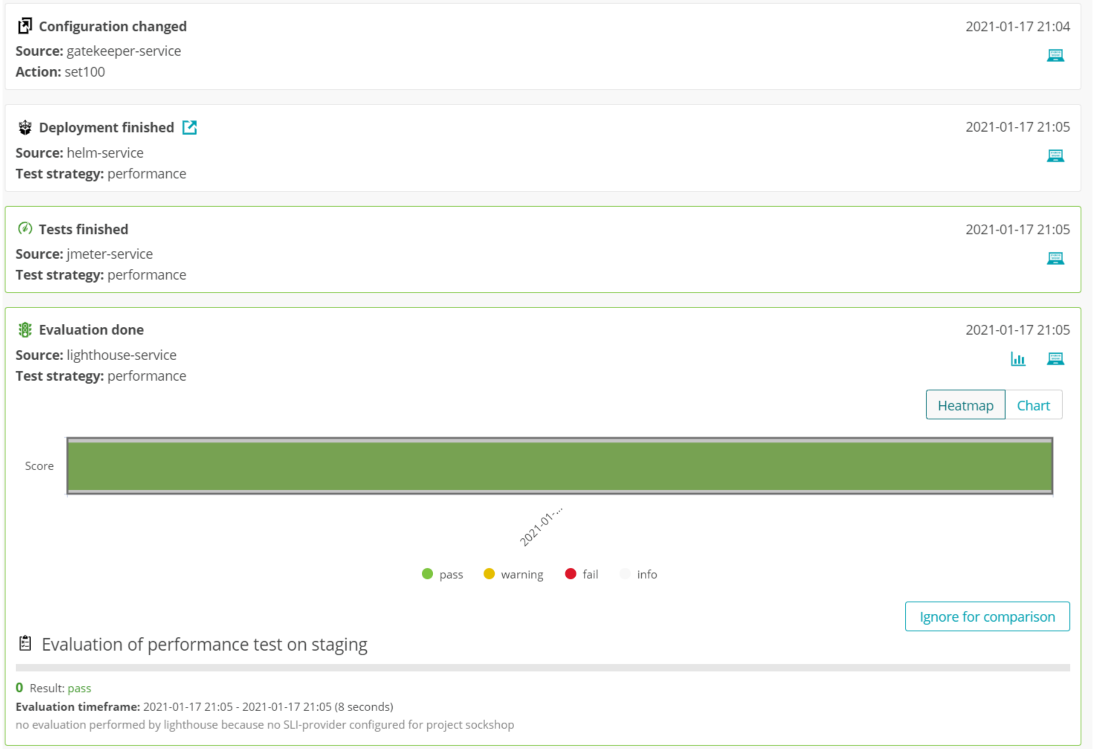
- Functional: basiccheck.jmx
- Performance: load.jmx

```
$ keptn add-resource --project=sockshop --stage=dev --service=carts --resource=jmeter/basiccheck.jmx ...
```

Deploy Artifact

```
$ keptn send event new-artifact --project=sockshop --service=carts --image=docker.io/keptnexamples/carts --tag=0.11.1
```





Configuration changed 2021-01-17 21:04
Source: gatekeeper-service
Action: set100

Deployment finished 2021-01-17 21:05
Source: helm-service
Test strategy: performance

Tests finished 2021-01-17 21:05
Source: jmeter-service
Test strategy: performance

Evaluation done 2021-01-17 21:05
Source: lighthouse-service
Test strategy: performance

Evaluation of performance test on staging

0 Result: pass
Evaluation timeframe: 2021-01-17 21:05 - 2021-01-17 21:05 (8 seconds)
no evaluation performed by lighthouse because no SLI-provider configured for project sockshop

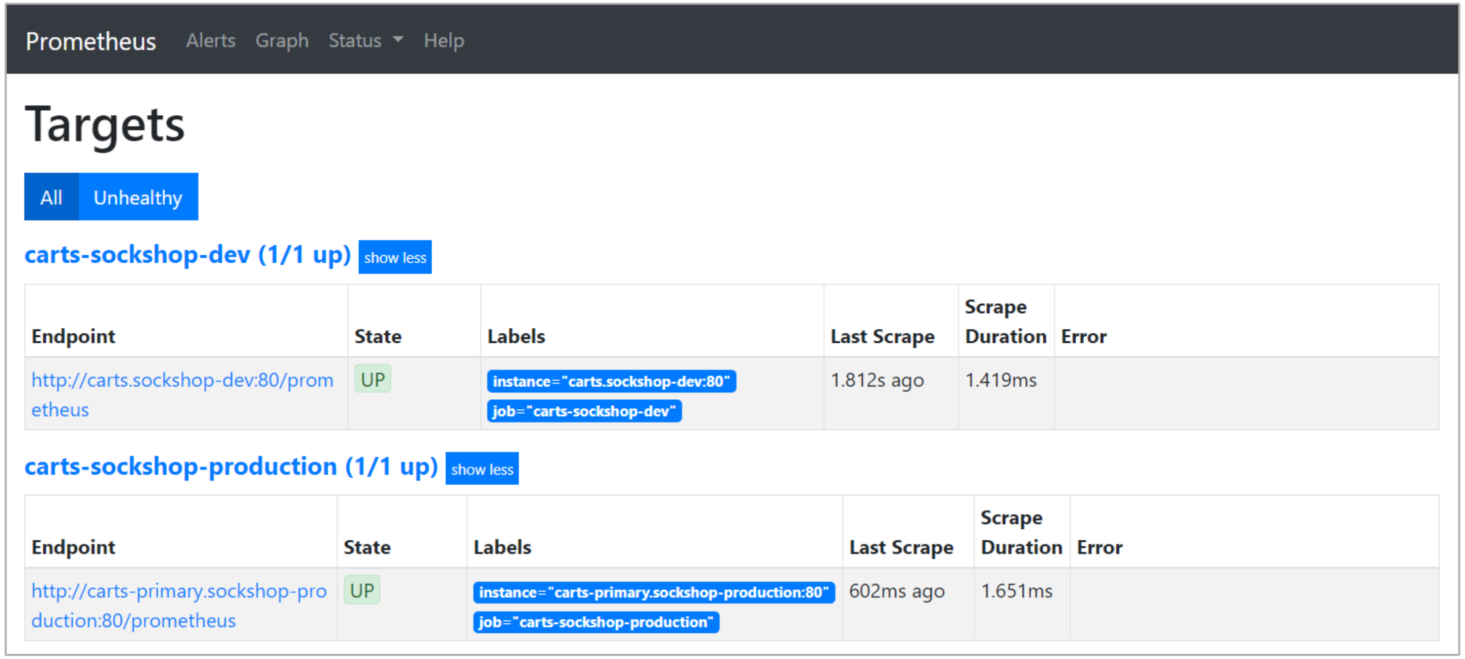
Add monitoring

Install prometheus-service

```
$ kubectl apply -f https://raw.githubusercontent.com/keptn-contrib/prometheus-service/release-0.3.6/deploy/service.yaml
```

Configure monitoring for carts service

```
$ keptn configure monitoring prometheus --project=sockshop --service=carts
```



Targets

carts-sockshop-dev (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://carts.sockshop-dev:80/prometheus	UP	instance="carts.sockshop-dev:80" job="carts-sockshop-dev"	1.812s ago	1.419ms	

carts-sockshop-production (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://carts-primary.sockshop-production:80/prometheus	UP	instance="carts-primary.sockshop-production:80" job="carts-sockshop-production"	602ms ago	1.651ms	

Learning from Google's SRE Practices

SLIs drive SLOs which inform SLAs

- Service Level Indicators (SLIs)
  - Definition: Measurable Metrics as the base for evaluation
  - Example: Error Rate of Login Requests
- Service Level Objectives (SLOs)
  - Definition: Binding targets for Service Level Indicators
  - Example: Login Error Rate must be less than 2% over a 30 day period
- Service Level Agreements (SLAs)
  - Definition: Business Agreement between consumer and provider typically based on SLO
  - Example: Logins must be reliable & fast (Error Rate, Response Time, Throughput) 99% within a 30 day window
- Google Cloud YouTube Video
  - SLIs, SLOs, SLAs, oh my! (class SRE implements DevOps): https://www.youtube.com/watch?v=tEylFyxbDLE

SLI/SLO-based evaluation implementation in Keptn

SLIs defined per SLI Provider as YAML

SLI Provider specific queries, e.g: Prometheus Metrics Query

```
indicators:
  error_rate:          "sum(rate(http_requests_total{job=’...’)"
  count_dbcalls:       "sum(rate(http_requests_total{job=’...’)"
  response_time_p95:   "histogram_quantile(0.95, sum(rate("
```

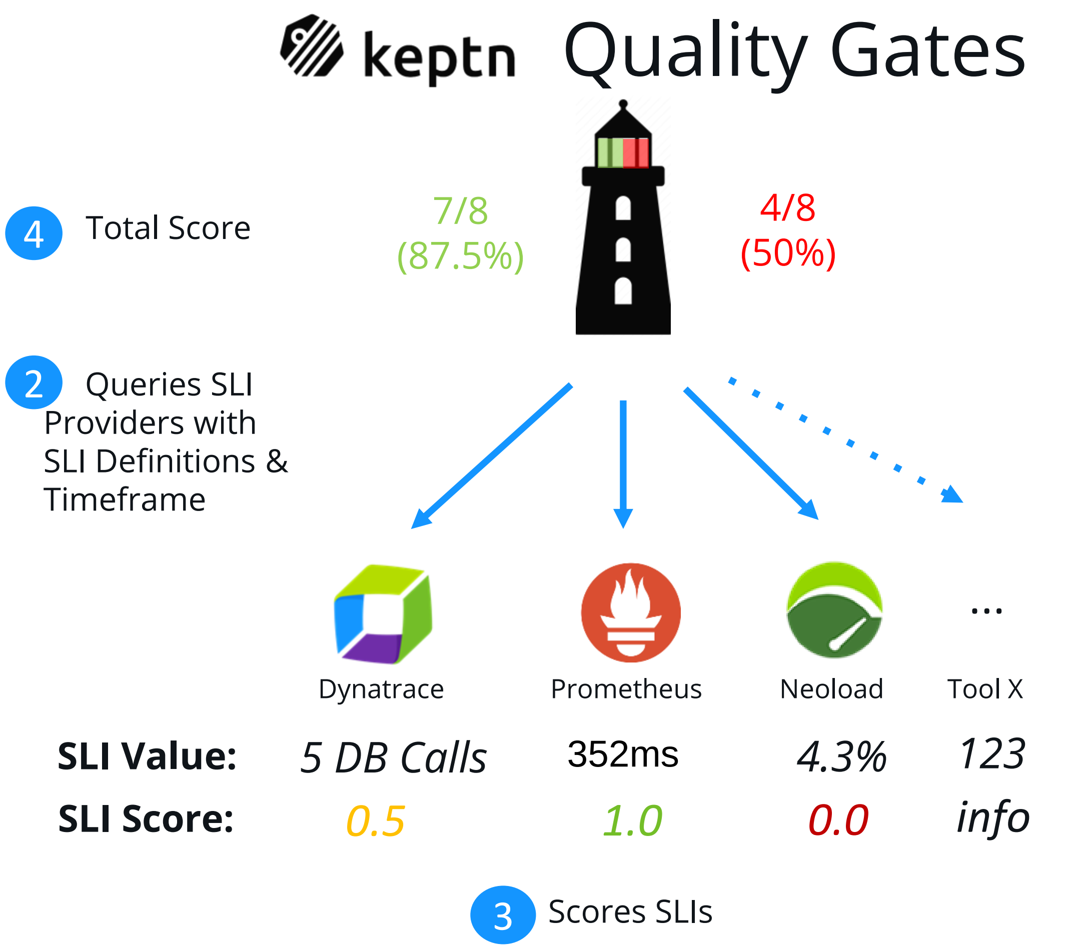
SLOs defined on Keptn Service Level as YAML

List of objectives with fixed or relative pass & warn criteria

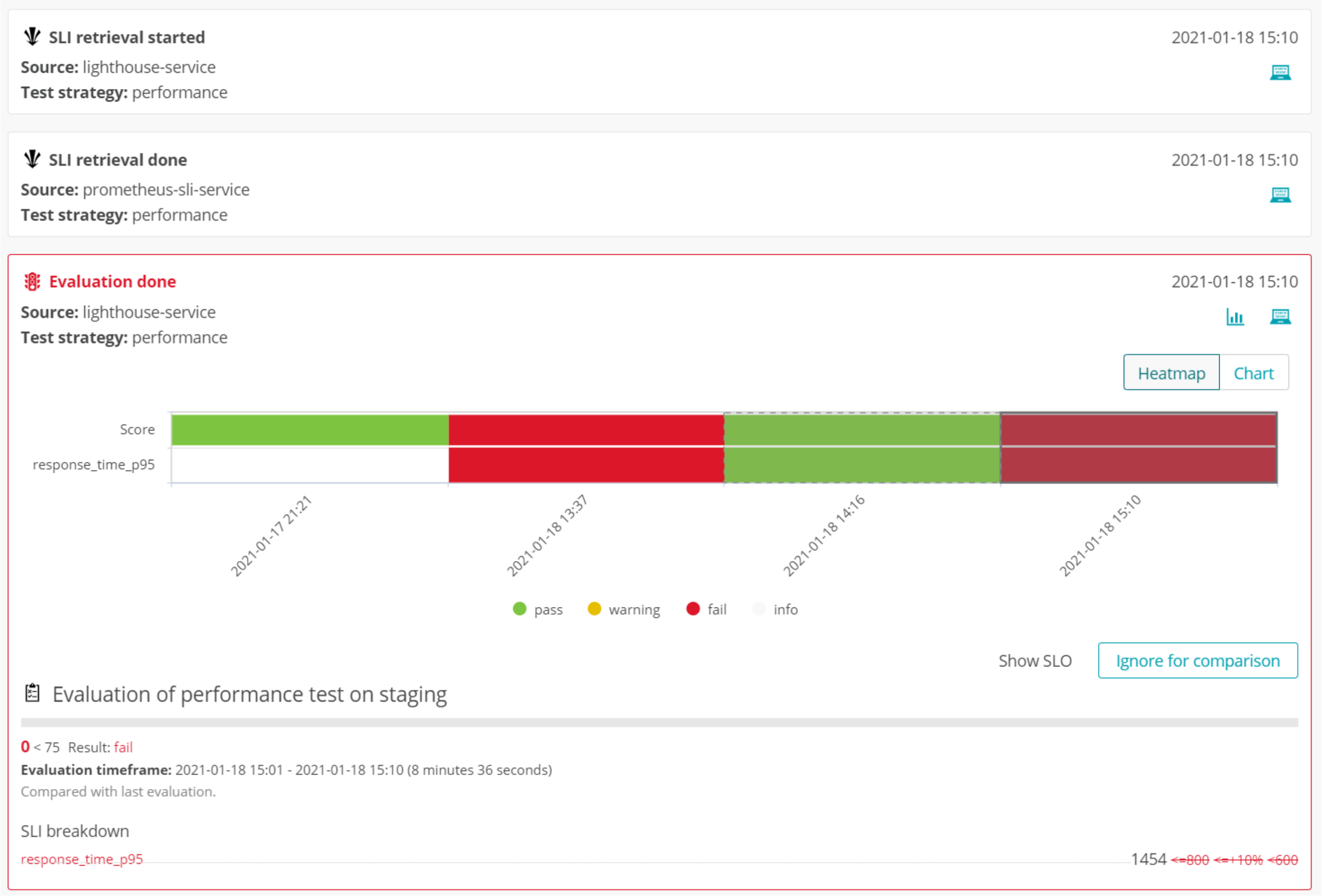
```
objectives:
  - sli: "response_time_p95"
    # pass if (relative change <= 10% AND absolute value is < 600ms)
    pass:
      - criteria:
          # relative values require a prefixed sign (plus or minus)
          - "<=+10%"
          # absolute values only require a logical operator
          - "<600"
    # if the response time is below 800ms, the result should be a wa
rning
    warning:
      - criteria:
          - "<=800"
    weight: 1
total_score:
  pass: "90%"
  warning: "75%"
```

1

```
$ keptn start-evaluation 30m myservice sli.yaml slo.yaml
```



SLO Evaluation



SLI retrieval started 2021-01-18 15:10

Source: lighthouse-service

Test strategy: performance

SLI retrieval done 2021-01-18 15:10

Source: prometheus-sli-service

Test strategy: performance

Evaluation done 2021-01-18 15:10

Source: lighthouse-service

Test strategy: performance

Evaluation of performance test on staging

0 < 75 Result: fail

Evaluation timeframe: 2021-01-18 15:01 - 2021-01-18 15:10 (8 minutes 36 seconds)

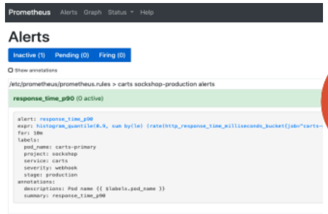
Compared with last evaluation.

SLI breakdown

response_time_p95 1454 ~~<=800~~ ~~<=+10%~~ ~~<600~~

Auto Remediation

Remediation in production based on SLOs



Alerts

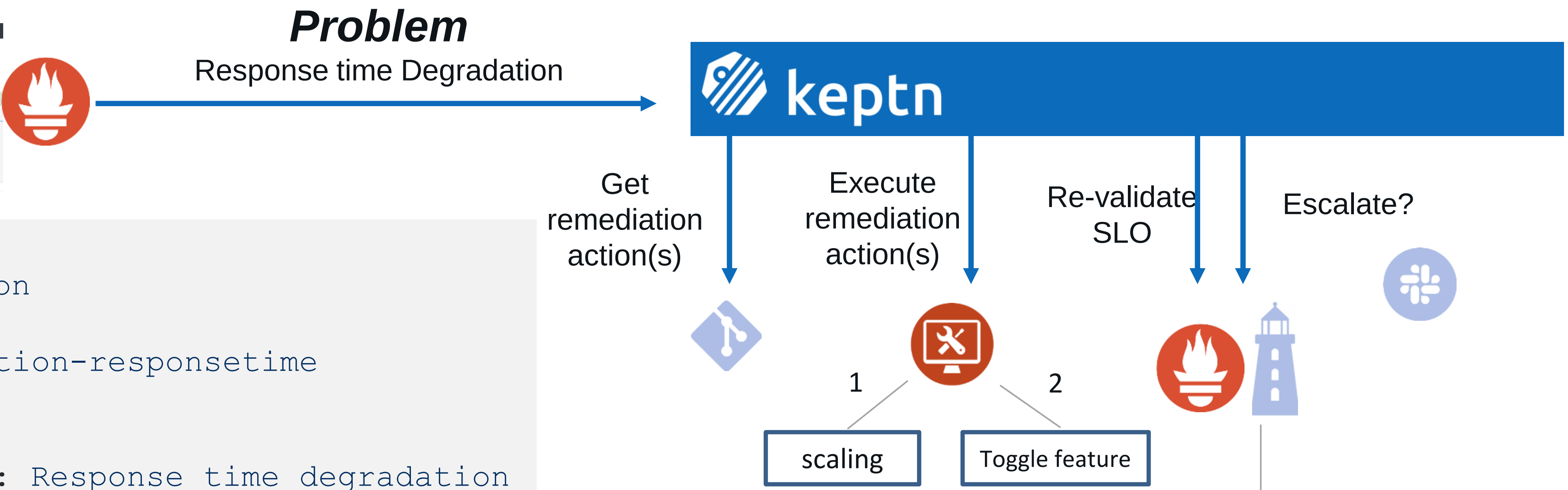
Inactive (1) Pending (0) Firing (0)

□ Show annotations

/etc/prometheus/prometheus.rules > carts sockshop-production alerts

response_time_p90 (0 active)

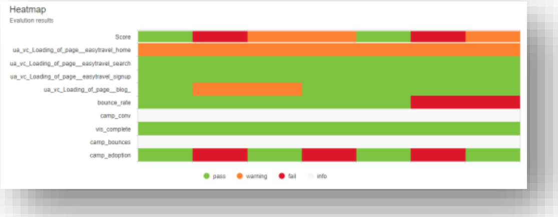
```
alert: response_time_p90
expr: histogram_quantile(0.9, sum by(le) (rate(http_response_time_milliseconds_bucket{job="carts-
for: 10m
labels:
  pod_name: carts-primary
  project: sockshop
  service: carts
  severity: webhook
  stage: production
annotations:
  descriptions: Pod name {{ $labels.pod_name }}
  summary: response_time_p90
```



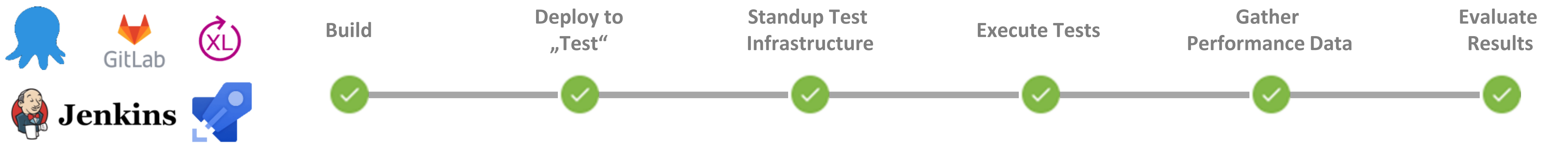
Problem

Response time Degradation

```
version: 0.2.0
kind: Remediation
metadata:
  name: remediation-responsetime
spec:
  remediations:
  - problemType: Response time degradation
    actionsOnOpen:
    - name: Scaling ReplicaSet by 1
1     action: scaling
      values:
        increment: +1
2   - name: Stop campaign
      action: toggle-feature
      values:
        status: disable
        campaign: $campaignid
```



Automated approval



Build
Deploy to „Test“ + Notify Keptn
Wait for Result

keptn

Test Scripts + Workload

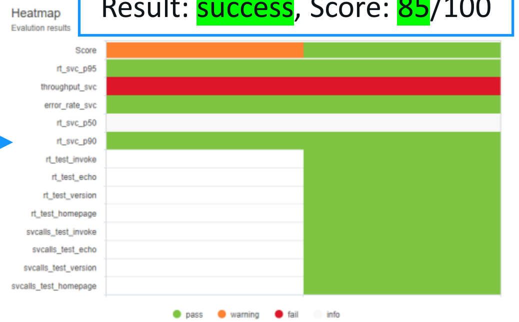
APACHE JMeter™

SLI & SLO

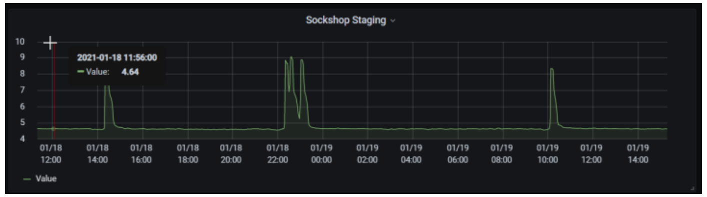
```
Rt(p95)  <  600ms
#ofSQLs  <=   5
cpu(max)<   80%
Java GC  <    2%
...
```

Validate SLOs

Result: success, Score: 85/100



Pull SLIs for Testing time frame



Wrap Up

Keptn: Data-Driven Delivery & Operations Automation

pick your use case

SLO-Quality Gates

Progressive Delivery

SRE Automation

Auto-Remediation

bring your configuration

You

(Dev/Ops/SRE)

shipyard

SLI/SLO

workload

runbook

connect your tools

...

keptn

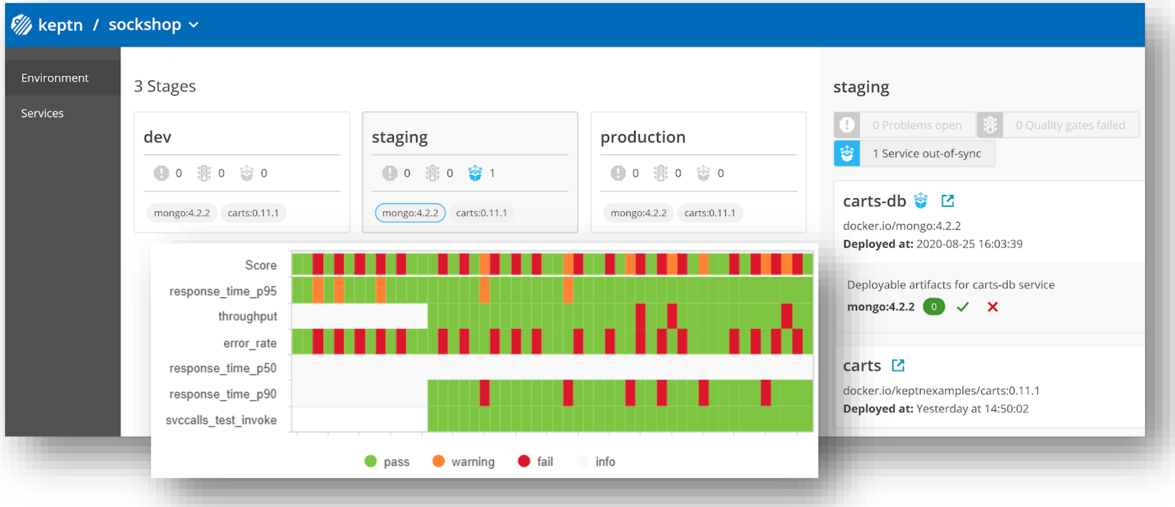
automates configuration and provides self-service for

Monitoring

Delivery

Reliability

Remediation



keptn / sockshop

Environment

Services

3 Stages

dev

0 0 0

mongo:4.2.2 carts:0.11.1

staging

0 0 1

mongo:4.2.2 carts:0.11.1

production

0 0 0

mongo:4.2.2 carts:0.11.1

staging

0 Problems open

0 Quality gates failed

1 Service out-of-sync

carts-db

docker.io/mongo:4.2.2

Deployed at: 2020-08-25 16:03:39

Deployable artifacts for carts-db service

mongo:4.2.2 0 ✓ ✗

carts

docker.io/keptnexamples/carts:0.11.1

Deployed at: Yesterday at 14:50:02

through event-driven process orchestration based on

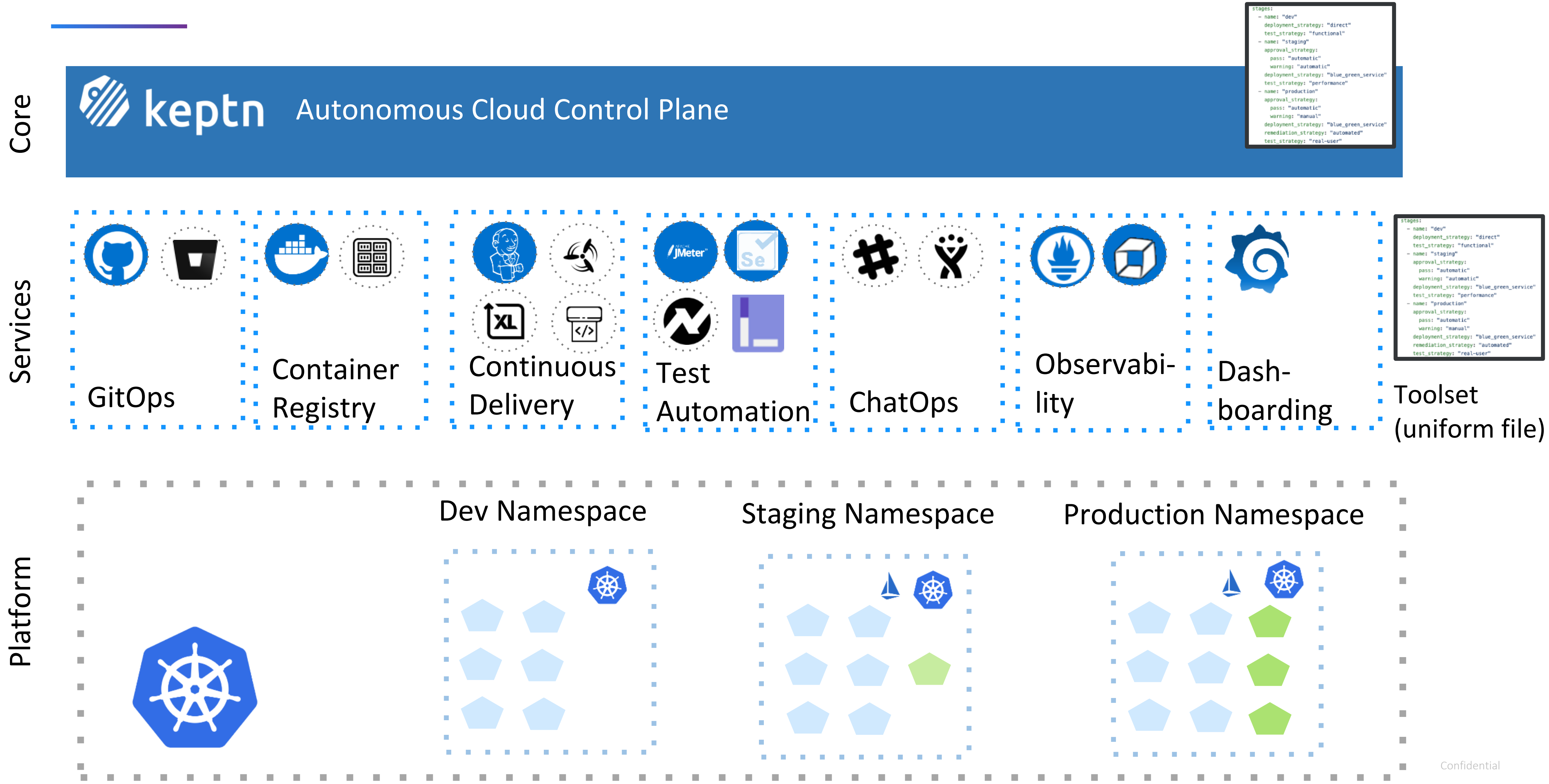
Declaration

GitOps

SLOs

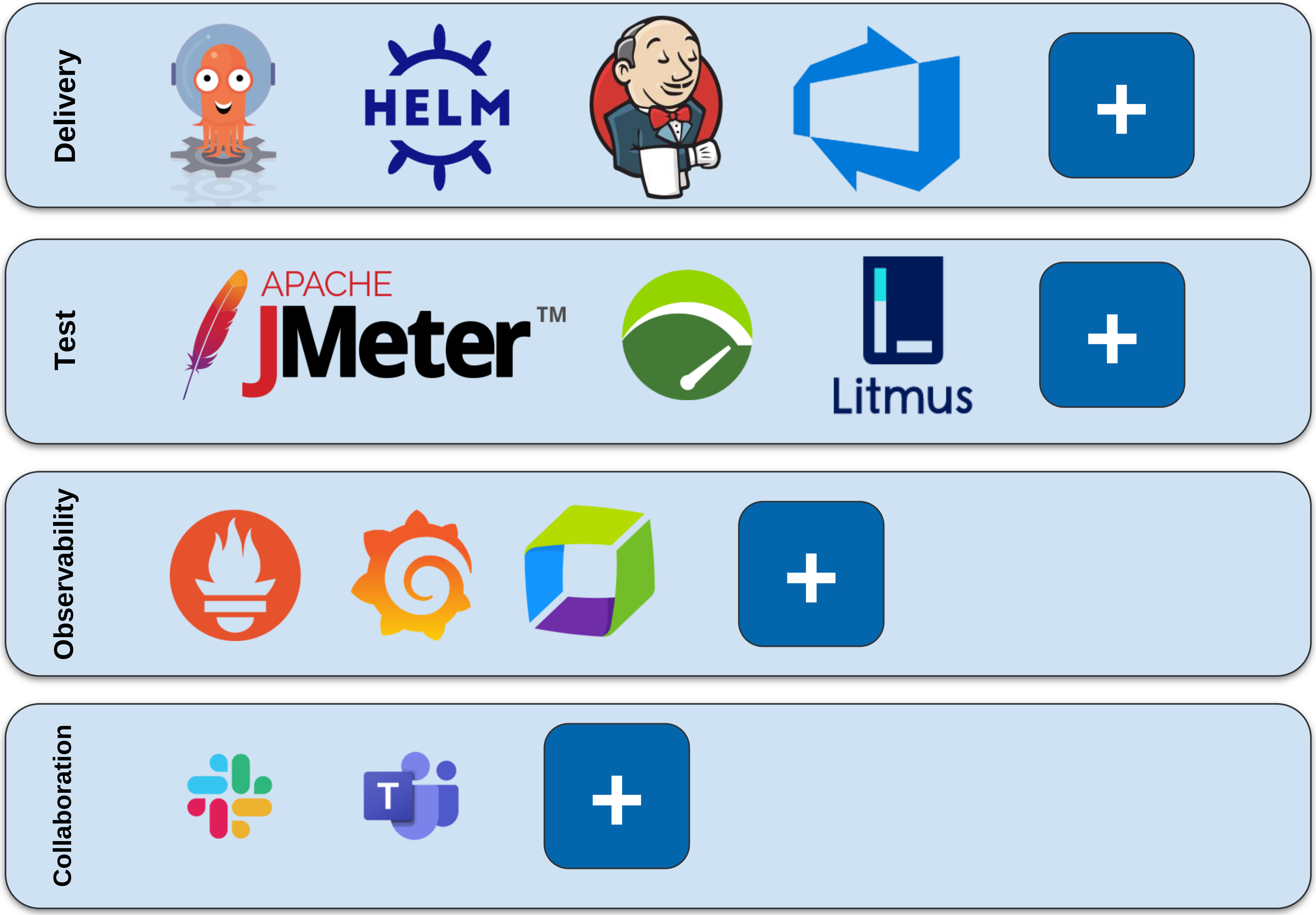
Standards

Keptn - conceptual architecture



```yaml
stages:
  - name: "dev"
    deployment_strategy: "direct"
    test_strategy: "functional"
  - name: "staging"
    approval_strategy:
      pass: "automatic"
      warning: "automatic"
    deployment_strategy: "blue_green_service"
    test_strategy: "performance"
  - name: "production"
    approval_strategy:
      pass: "automatic"
      warning: "manual"
    deployment_strategy: "blue_green_service"
    remediation_strategy: "automated"
    test_strategy: "real-user"
```

```yaml
stages:
  - name: "dev"
    deployment_strategy: "direct"
    test_strategy: "functional"
  - name: "staging"
    approval_strategy:
      pass: "automatic"
      warning: "automatic"
    deployment_strategy: "blue_green_service"
    test_strategy: "performance"
  - name: "production"
    approval_strategy:
      pass: "automatic"
      warning: "manual"
    deployment_strategy: "blue_green_service"
    remediation_strategy: "automated"
    test_strategy: "real-user"
```



Delivery

Test

Observability

Collaboration

… and growing!

Thank you!

Robin Wyss
Sales Engineer at Dynatrace

https://www.linkedin.com/in/robinwyss
@robinwyss

Web	http://keptn.sh/
Twitter	@keptnProject
GitHub	https://github.com/keptn/keptn
Tutorials	https://tutorials.keptn.sh
Slack	http://slack.keptn.sh