Performance Testing in Keptn Using K6

Jainam Shah

Brief about me!

- Software Engineer *@JioSaavn*
- GSoC *@Keptn*
- Loves Hackathons
- Avid Traveller
- Enjoys playing sports

Keptn

What is Keptn?

- Open-source control plane for continuous delivery and automated operations
- Streamlines deployment and management of cloud-native applications

Keptn Features

- Declarative approach: GitOps principles, version-controlled repositories
- Event-driven automation: Trigger actions based on events
- Multi-stage pipelines: Progressive rollouts, canary releases
- Policy-based quality gates: Enforce stability and reliability
- Observability and monitoring: Integration with monitoring tools, auto-remediation

Keptn Lifecycle Toolkit

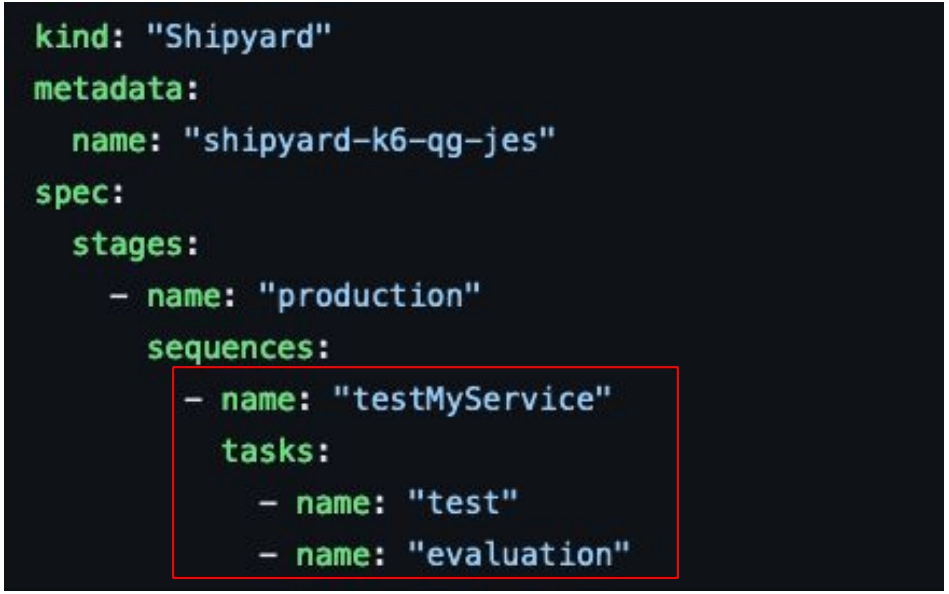
- The Keptn Lifecycle Toolkit is a tool that helps cloud-native teams manage the lifecycle of their applications
- It provides a number of features, including:
  - Pre- and post-deployment checks
  - Application health checks

- We'll be focusing on Keptn Integrations triggered via CloudEvents in this talk

Declarative Multi-Stage Delivery

- Keptn allows you to define a multi-stage delivery workflow declaratively
- ***Shipyard*** file defines the task instance for delivery

Shipyard File



```yaml
kind: "Shipyard"
metadata:
  name: "shipyard-k6-qg-jes"
spec:
  stages:
    - name: "production"
      sequences:
        - name: "testMyService"
          tasks:
            - name: "test"
            - name: "evaluation"
```

K6

Performance Testing

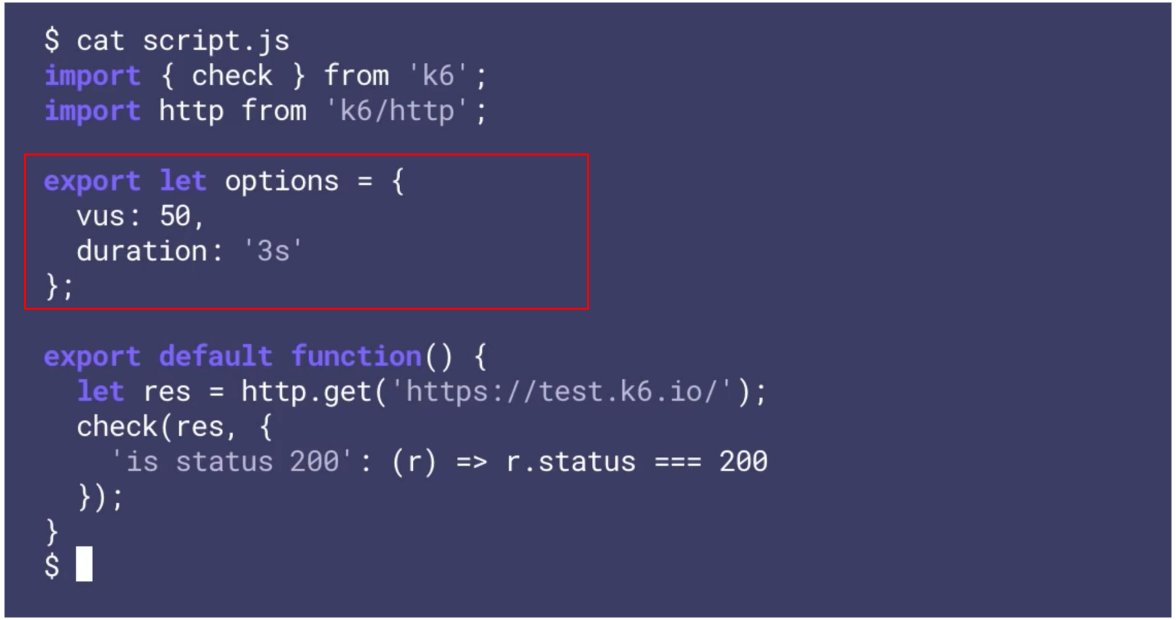
Performance testing is required for several reasons:

1. Performance Optimization
2. User Experience
3. Scalability and Capacity Planning
4. Stability and Reliability
5. Compliance and SLA Validation

What is K6?

- K6 is an open-source tool widely used for load testing
- K6 has been widely accepted across cloud-native tools
- K6 has a very good resource utilization with one load generator simulating tens of thousands of virtual users
- K6 is code driven and uses JavaScript as scripting language
- K6 has native support for Prometheus and other data tools using extensions

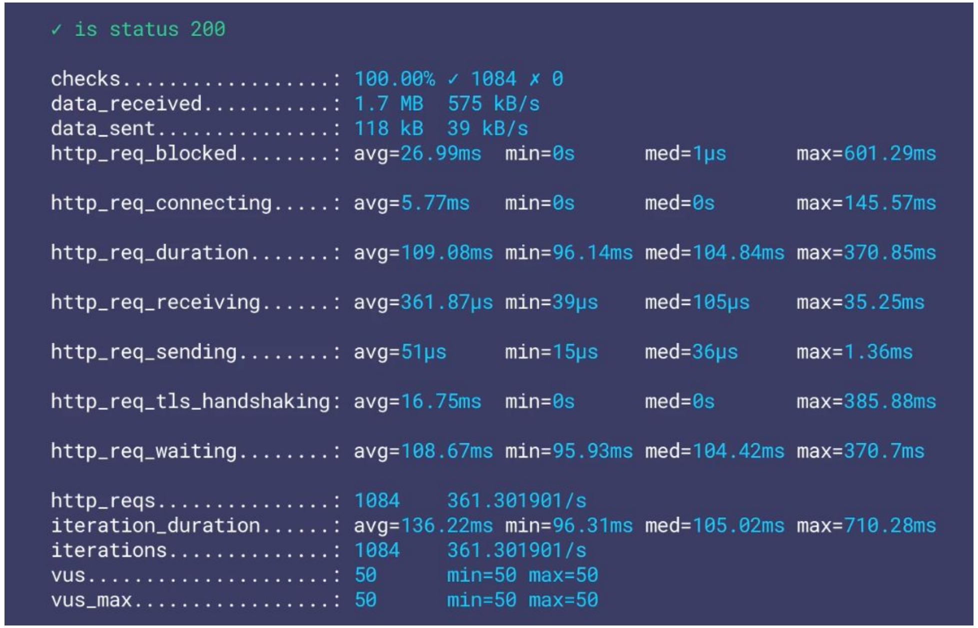
K6 Script



```
$ cat script.js
import { check } from 'k6';
import http from 'k6/http';

export let options = {
  vus: 50,
  duration: '3s'
};

export default function() {
  let res = http.get('https://test.k6.io/');
  check(res, {
    'is status 200': (r) => r.status === 200
  });
}
$ 
```

Image Source: K6 docs

K6 Run



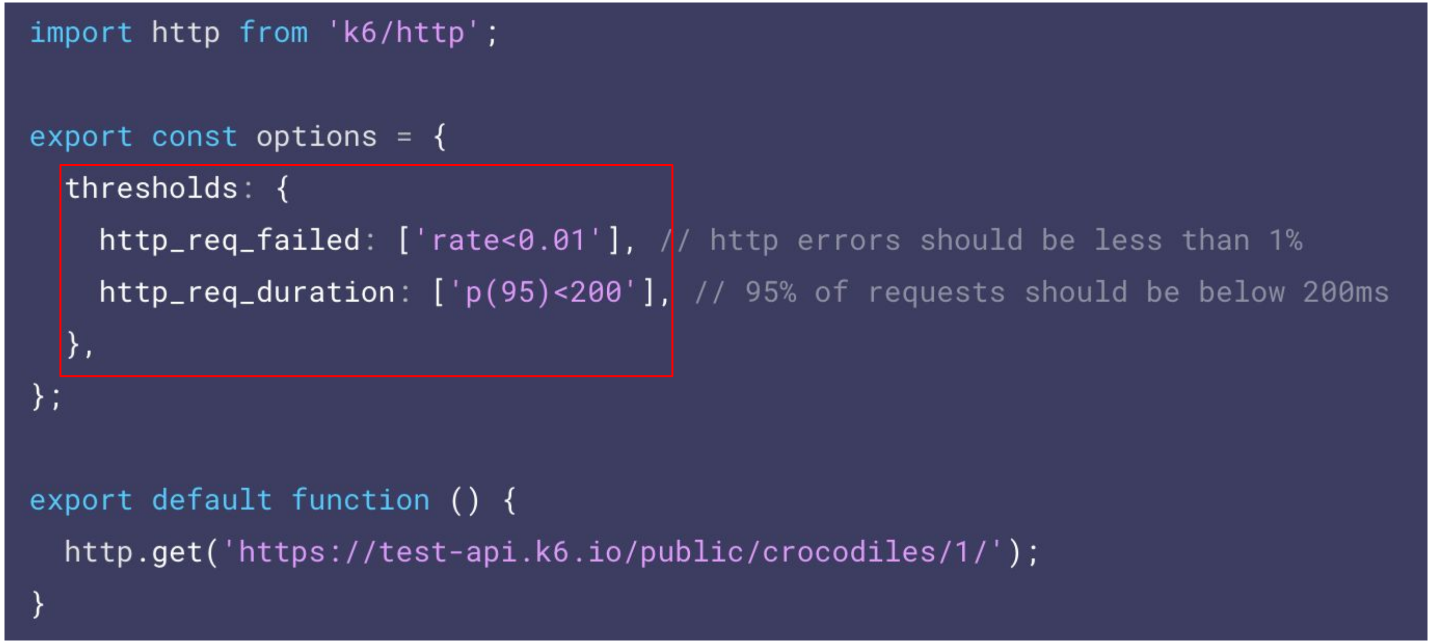
```
✓ is status 200

checks...................: 100.00% ✓ 1084 ✗ 0
data_received............: 1.7 MB  575 kB/s
data_sent................: 118 kB  39 kB/s
http_req_blocked.........: avg=26.99ms  min=0s      med=1µs      max=601.29ms

http_req_connecting......: avg=5.77ms   min=0s      med=0s       max=145.57ms

http_req_duration........: avg=109.08ms min=96.14ms med=104.84ms max=370.85ms

http_req_receiving.......: avg=361.87µs min=39µs    med=105µs    max=35.25ms

http_req_sending.........: avg=51µs     min=15µs    med=36µs     max=1.36ms

http_req_tls_handshaking: avg=16.75ms  min=0s      med=0s       max=385.88ms

http_req_waiting.........: avg=108.67ms min=95.93ms med=104.42ms max=370.7ms

http_reqs................: 1084    361.301901/s
iteration_duration.......: avg=136.22ms min=96.31ms med=105.02ms max=710.28ms
iterations...............: 1084    361.301901/s
vus......................: 50      min=50 max=50
vus_max..................: 50      min=50 max=50
```

Image Source: K6 docs

K6 Thresholds

- *Thresholds* are pass/fail criteria for your test metrics
- If any test metrics fails, then K6 returns with a non-zero exit code
- K6 uses thresholds to codify their SLOs
- Some examples look like
  - Less than 1% of requests return an error
  - 95% of requests have a response time below 200ms
  - A specific endpoint always responds within 300ms

K6 Thresholds Script



```
import http from 'k6/http';

export const options = {
  thresholds: {
    http_req_failed: ['rate<0.01'], // http errors should be less than 1%
    http_req_duration: ['p(95)<200'], // 95% of requests should be below 200ms
  },
};

export default function () {
  http.get('https://test-api.k6.io/public/crocodiles/1/');
}
```

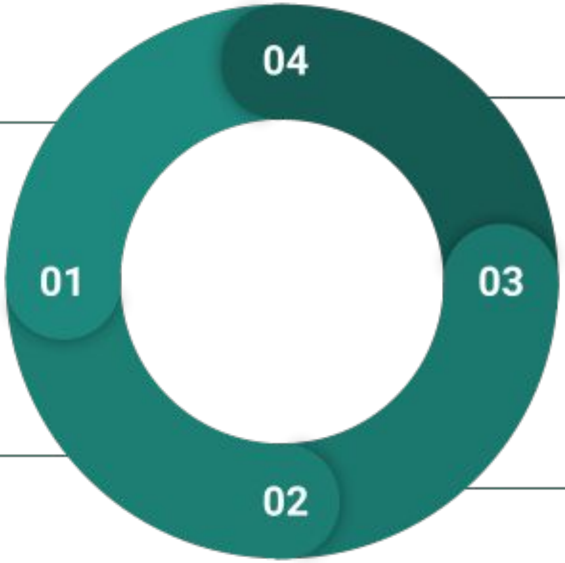
Image Source: K6 docs

K6 Extensions

- K6 Extensions help expand the potential use cases of K6
- It allows user to build a custom K6 binary and use it to write across other Platforms like Prometheus and Dynatrace
- ***XK6*** is command-line tool and framework written in Go used for building custom K6 binary

K6 Integration in Keptn :)

Keptn-K6 Workflow



Declare Shipyard

A shipyard is defined at the level of a project. It defines the stages each deployment has to go through until release in final stage, e.g., ***dev & production stage***

Sequences in Stage

Sequences can be added to a stage. A sequence is an ordered list of tasks that are triggered sequentially. It consists of array of *Tasks* and *triggeredOn* properties, e.g., ***test*** task will be used in this tutorial

Create a Service

For any microservice which needs to be tested and deployed, we'll add config file for JES along with K6 testing files. The config file will have *K6 run command* and it'll ***pass or failure*** based on *K6 Thresholds*

Setup Job Executor Service

After setting up JES, we'll add subscription to ***test.triggered*** CloudEvent in *Settings*. So, whenever test task is called upon, it'll use the config for JES and execute test. Here we'll use K6 docker image in config

Keptn Integration Flow

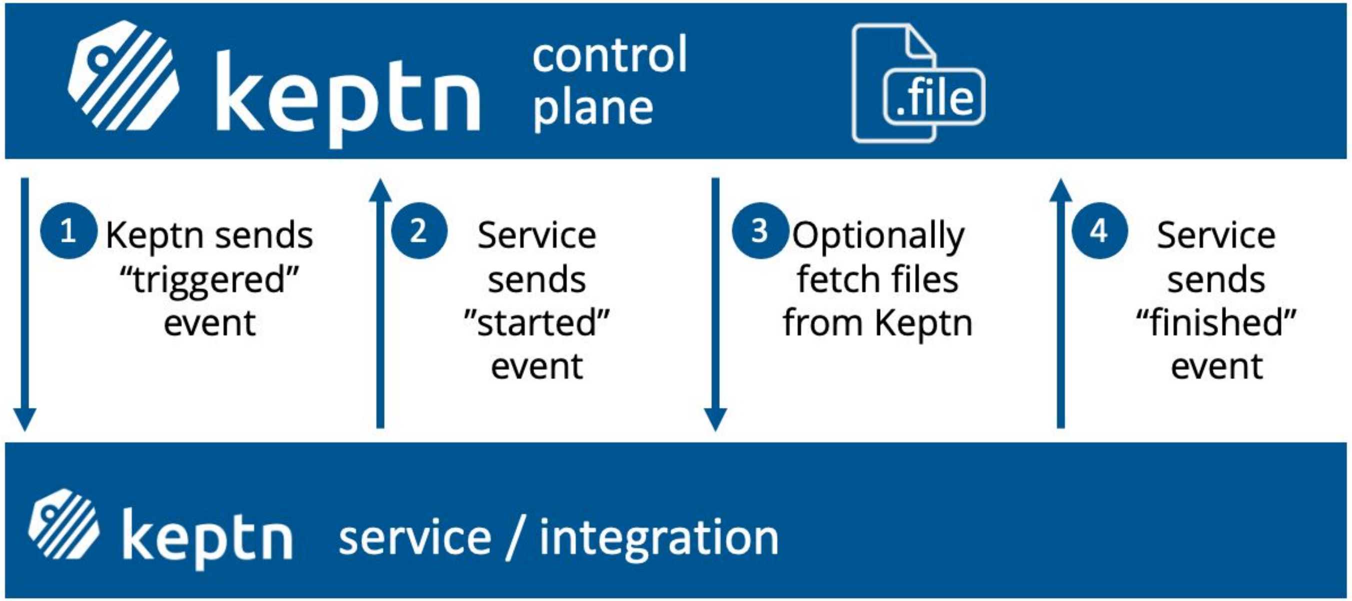
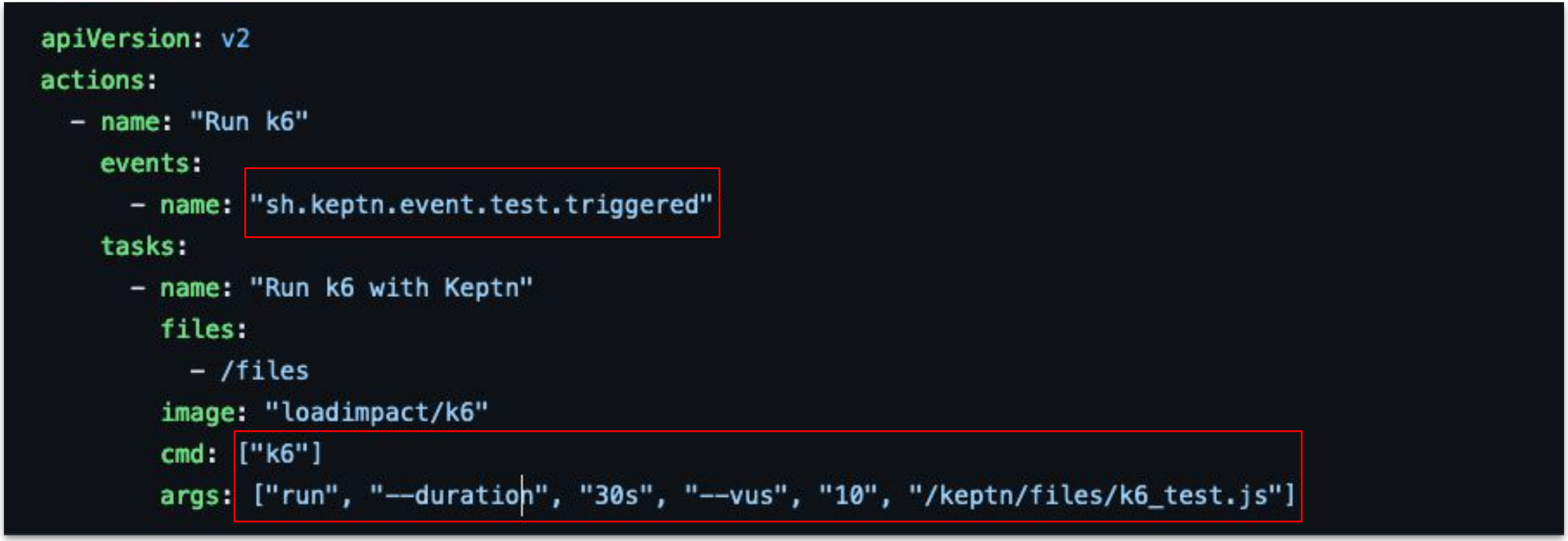

keptn control plane
.file
1 Keptn sends "triggered" event
2 Service sends "started" event
3 Optionally fetch files from Keptn
4 Service sends "finished" event
keptn service / integration

Image Source: Keptn docs

Job Executor Service

- A single service which provides the means to run any workload orchestrated by Keptn
- This service can execute any framework with just a few lines of YAML configuration
- No need to write or maintain any new code

Job Executor Service (config.yaml)

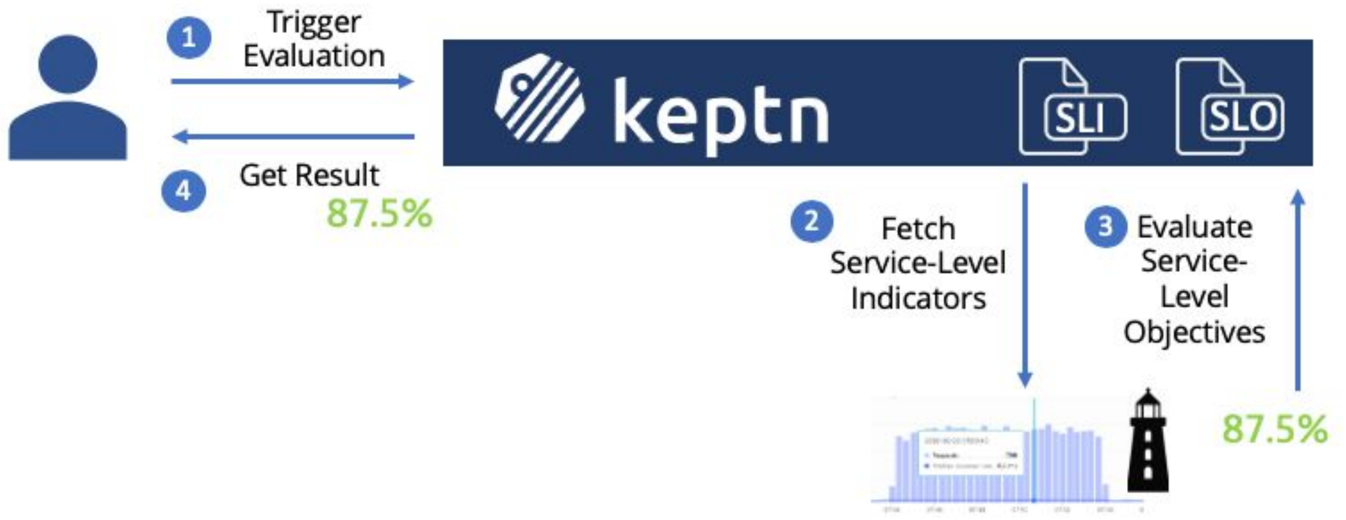


```yaml
apiVersion: v2
actions:
  - name: "Run k6"
    events:
      - name: "sh.keptn.event.test.triggered"
    tasks:
      - name: "Run k6 with Keptn"
        files:
          - /files
        image: "loadimpact/k6"
        cmd: ["k6"]
        args: ["run", "--duration", "30s", "--vus", "10", "/keptn/files/k6_test.js"]
```

Quality Gates

- Keptn quality gates are automated checks that ensure the quality of a software release before it is **promoted to production**

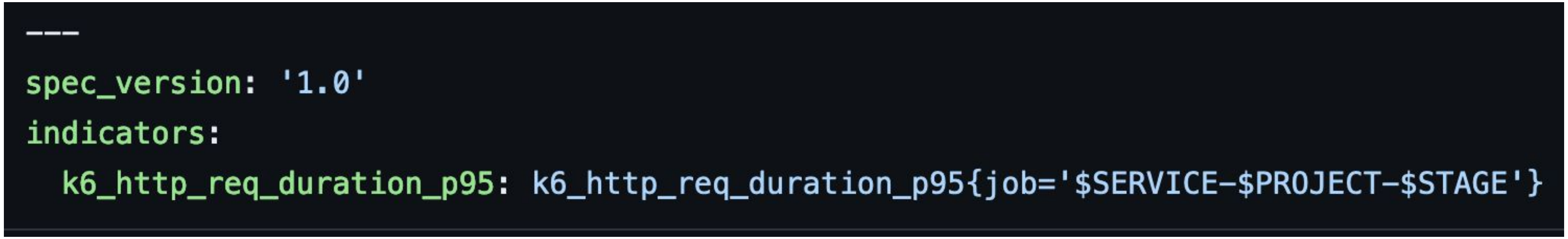
Quality Gates



1 Trigger Evaluation
4 Get Result
87.5%
keptn
SLI
SLO
2 Fetch Service-Level Indicators
3 Evaluate Service-Level Objectives
87.5%

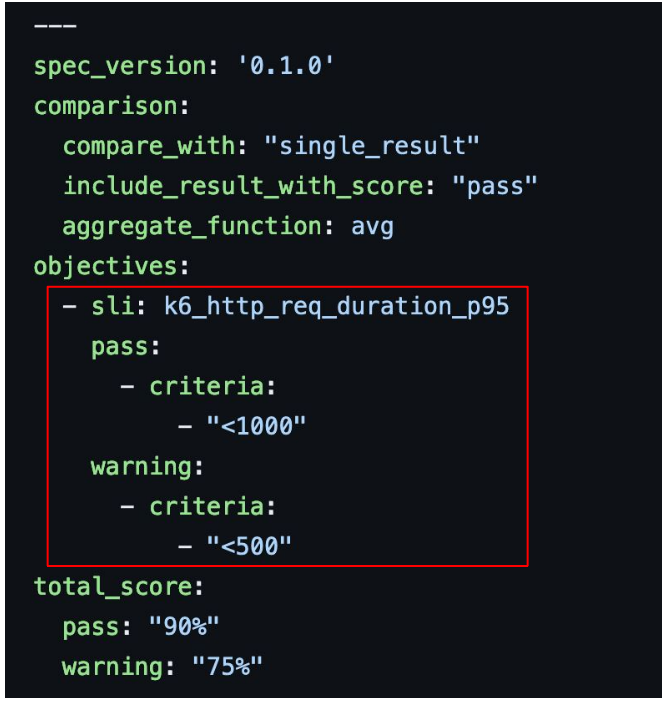
Image Source: Keptn docs

Quality Gates (sli.yaml)



```yaml
---
spec_version: '1.0'
indicators:
  k6_http_req_duration_p95: k6_http_req_duration_p95{job='$SERVICE-$PROJECT-$STAGE'}
```

Quality Gates (slo.yaml)



```yaml
---
spec_version: '0.1.0'
comparison:
  compare_with: "single_result"
  include_result_with_score: "pass"
  aggregate_function: avg
objectives:
  - sli: k6_http_req_duration_p95
    pass:
      - criteria:
          - "<1000"
    warning:
      - criteria:
          - "<500"
total_score:
  pass: "90%"
  warning: "75%"
```

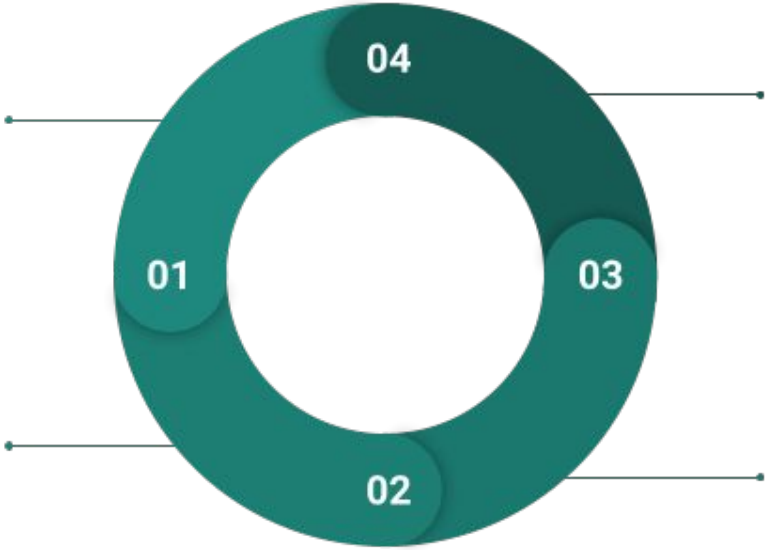
K6 - Quality Gates Workflow

Evaluation Task in Shipyard

Evaluation is a reserved task in Keptn. It'll be used for quality gate evaluation. We'll add it to our *Sequence* after ***test*** task to automatically trigger SLO compliance after successful performance testing.

K6 writes to Prometheus

Using ***K6-Prometheus extension***, we'll write the metrics to Prometheus. We'll add a tag based on Keptn Project, Stage and Service as a unique identifier of metrics. ***Prometheus Service*** of Keptn will fetch these metrics.



Quality Gate Evaluation

Once we have the SLI values of the test performed, LightHouse service will use the ***SLO config*** to for quality gate. SLO config defines the pass criteria for the SLI values

Get SLI

Lighthouse Service of Keptn is responsible for Quality Gates Evaluation. This service internally calls ***get-sli.triggered*** CloudEvent. SLI provider service (Prometheus Service in this case) will receive this event and fetch the metrics. ***Prometheus Expressions*** will be used to query mentioned in SLI config

Demo

Summary

- Declarative Multi Stage Pipeline
- Job Executor Service
  - K6 Thresholds for SLO validation
- Quality Gates
  - Several SLOs that are evaluated and scored

Thank You!

- Any Questions?
- Tutorial link for this talk →

@jainammm

@jvenommm