Reproducible Benchmarking of Cloud-Native Applications with the Kubernetes Operator Pattern

Sören Henning, Benedikt Wetzel and Wilhelm Hasselbring
Cloud-native Applications in Kubernetes
Cloud-native Applications in Kubernetes

creates

- my-deployment.yaml
- my-service.yaml
- ...

...
Cloud-native Applications in Kubernetes

> kubectl apply -f ______

`my-deployment.yaml`
`my-service.yaml`
...

creates
Cloud-native Applications in Kubernetes

```
kubectl apply -f ______
```

creates

```
my-deployment.yaml
my-service.yaml
...
```

forwards to

```
> kubectl apply -f ______
```

API

Kubernetes
Cloud-native Applications in Kubernetes

```
kubectl apply -f __________
```

- `kubectl apply -f` command is used to apply configuration files to Kubernetes.
- `my-deployment.yaml`
- `my-service.yaml`
- `...`

- The `kubectl apply` command creates objects from the specified YAML files.
- It forwards the configuration to the Kubernetes API and the kubelets.
- The API orchestrates the creation of the application components.
Benchmarks Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
Benchmarking Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```
Benchmarking Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

where to run?
another version?
Benchmarks Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

where to run?

another version?

async operation
Benchmarking Cloud-native Applications: The Classical Way

start-my-benchmark.sh

```
helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

where to run?

another version?

async operation

custom config?
Benchmarking Cloud-native Applications: The Classical Way

```bash
helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

where to run?

another version?

async operation

custom config?

Reproducibility?
Verifiability?
Usability?
Benchmarking Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

where to run?

another version?

async operation

custom config?

Reproducibility?
Verifiability?
Usability?

```
Benchmarking Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

**Suggestion:**
Use established cloud-native tools
Benchmarks cloud-native applications: the classical way

**start-my-benchmark.sh**

```
helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...```

Suggestion:
Use established cloud-native tools.
Benchmarking Cloud-native Applications: The Classical Way

```
start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...
```

Suggestion:
Use established cloud-native tools
Benchmarking Cloud-native Applications: The Classical Way

Suggestion:
Use established cloud-native tools

start-my-benchmark.sh

```
helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...```
Benchmarking Cloud-native Applications: The Classical Way

start-my-benchmark.sh

helm install my-middleware -f mw-config.yaml
helm install my-monitoring-tool -f mon-cfg.yaml
...
kubectl apply -f load-generator-config.yaml
kubectl apply -f load-generator-deploy.yaml
...
kubectl apply -f sut-xyz-config.yaml
kubectl apply -f sut-xyz-deploy.yaml
kubectl apply -f sut-xyz-svc.yaml
...
./collect-monitoring-data.sh
...
kubectl delete deployment load-generator
...

Suggestion:
Use established cloud-native tools

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes Prometheus HELM

Operator Pattern
The Kubernetes Operator Pattern

> kubectl apply -f ```my-custom-resource.yaml```
The Kubernetes Operator Pattern

> kubectl apply -f ______

creates

my-custom-resource.yaml

forwards to

Operator

reconciles

notifies

updates state

orchestrates
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer

Benchmark Marker
Custom Resource Definitions (CRDs) for Benchmarking

- Expert regarding specific application types or software services e.g., researchers, engineers, standardization committees
- Defines how metrics and results can be interpreted to compare different SUTs
- Bundles all of this in a benchmarking tool or for a benchmarking tool
Custom Resource Definitions (CRDs) for Benchmarking

**Benchmark Designer**
- Expert regarding specific application types or software services e.g., researchers, engineers, standardization committees
- Defines how metrics and results can be interpreted to compare different SUTs
- Bundles all of this in a benchmarking tool or for a benchmarking tool

**Benchmarker**
- Compares and ranks different existing SUTs
- Evaluates new methods tools against a defined standard
- Repeats previous experiments
- Executes existing benchmarks
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer creates Benchmark

Benchmark Designer

Benchmark

Benchmarker
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer creates Benchmark, which is published to the Benchmark Repository.

Benchmarker
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer creates Benchmark, which refers to Execution. Benchmark retrieves from Benchmark Repository and publishes to Benchmark Designer. Benchmark Designer creates CRDs.
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer

Benchmark

Execution

Benchmarking Operator

Benchmark Repository

creates

deploys to

publishes to

retrieves from

creates

deploys to

refers to
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer

Benchmark

Benchmark Operator

Execution

Research Data Repository

Benchmark Repository
Custom Resource Definitions (CRDs) for Benchmarking

Benchmark Designer

Benchmark

Execution

Benchmarker

Benchmarking Operator

Benchmark Repository

Research Data Repository
Architecture for a Benchmarking Operator

- **Orchestration API**: Deploys, notifies, updates state.
- **Benchmarker**: Observes, registers CRDs.
- **Package Manager**: Issues installation, installs.
- **Results**: Stores, queries.
- **Visualization**: Queries.
- **Monitoring**: Queries, monitors.
- **Load Generator**: Starts/stops.
- **Microservice (SUT)**: Starts/stops.

**Key Terms**:
- **CRDs**: Custom Resource Definitions
- **YAML**: YAML is a human- and machine-readable data format.
Scalability Benchmarking with Theodolite

- **load intensity**: e.g., messages per second, size of messages, ...
- **resources**: e.g., pods, CPU cores per pod, ...

- **min. resources to handle load**: i.e., not violation of SLOs

https://github.com/cau-se/theodolite
Theodolite Benchmarks

apiVersion: theodolite.com/v1
kind: benchmark
metadata:
  name: ucl-kstreams
spec:
  appResource:
    - "ucl-kstreams-deployment.yaml"
    - "aggregation-service.yaml"
    - "jmx-configmap.yaml"
    - "ucl-service-monitor.yaml"
  loadGenResource:
    - "ucl-load-generator-deployment.yaml"
    - "ucl-load-generator-service.yaml"
  resourceTypes:
    - typeName: "Instances"
      patchers:
        - type: "ReplicaPatcher"
          resource: "ucl-kstreams-deployment.yaml"
  loadTypes:
    - typeName: "NumSensors"
      patchers:
        - type: "EnvVarPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            variableName: "NUM_SENSORS"
            container: "workload-generator"
        - type: "NumSensorLoadGeneratorReplicaPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            loadGenMaxRecords: "15000"
  kafkaConfig:
    bootstrapServer: "theodolite-cp-kafka:9092"
    topics:
      - name: "input"
        numPartitions: 40
        replicationFactor: 1
        - name: "theodolite-.*"
          removeOnly: True
Theodolite **Benchmarks**

Kubernetes resources for deploying the benchmark

```yaml
apiVersion: theodolite.com/v1
kind: benchmark
metadata:
  name: ucl-kstreams
spec:
  appResource:
    - "ucl-kstreams-deployment.yaml"
    - "aggregation-service.yaml"
    - "jmx-configmap.yaml"
    - "ucl-service-monitor.yaml"
  loadGenResource:
    - "ucl-load-generator-deployment.yaml"
    - "ucl-load-generator-service.yaml"
  resourceTypes:
    - type: "Instances"
      patchers:
        - type: "ReplicaPatcher"
          resource: "ucl-kstreams-deployment.yaml"
  loadTypes:
    - type: "NumSensors"
      patchers:
        - type: "EnvVarPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            variableName: "NUM_SENSORS"
            container: "workload-generator"
        - type: "NumSensorLoadGeneratorReplicaPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            loadGenMaxRecords: "15000"
  kafkaConfig:
    bootstrapServer: "theodolite-cp-kafka:9092"
    topics:
      - name: "input"
        numPartitions: 40
        replicationFactor: 1
        - name: "theodolite-.*"
          removeOnly: True
```
Theodolite **Benchmarks**

![Diagram showing the relationship between load intensity and resources]
Theodolite **Benchmarks**

```
apiVersion: theodolite.com/v1
kind: benchmark
metadata:
  name: ucl-kstreams
spec:
  appResource:
    - "ucl-kstreams-deployment.yaml"
    - "aggregation-service.yaml"
    - "jmx-configmap.yaml"
    - "ucl-service-monitor.yaml"
  loadGenResource:
    - "ucl-load-generator-deployment.yaml"
    - "ucl-load-generator-service.yaml"
  resourceTypes:
    - typeName: "Instances"
      patchers:
        - type: "ReplicaPatcher"
          resource: "ucl-kstreams-deployment.yaml"
  loadTypes:
    - typeName: "NumSensors"
      patchers:
        - type: "EnvVarPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            variableName: "NUM_SENSORS"
            container: "workload-generator"
        - type: "NumSensorLoadGeneratorReplicaPatcher"
          resource: "ucl-load-generator-deployment.yaml"
          properties:
            loadGenMaxRecords: "15000"
  kafkaConfig:
    bootstrapServer: "theodolite-cp-kafka:9092"
  topics:
    - name: "input"
      numPartitions: 40
      replicationFactor: 1
    - name: "theodolite-.*"
      removeOnly: True
```
Theodolite **Benchmarks**

```
$ kubectl get benchmarks
NAME                        AGE
osp-flink                   14d
osp-kstreams                14d
osp-spark-streaming         11d
theodolite-uc2-flink       16d
theodolite-uc2-kstreams    16d
theodolite-uc3-flink       16d
theodolite-uc3-kstreams    16d
```
Theodolite *Executions*

Benchmark

```
apiVersion: theodolite.com/v1
kind: execution
metadata:
  name: theodolite-example-execution
spec:
  benchmark: "uc1-kstreams"
  load:
    loadType: "NumSensors"
    loadValues: [25000, 50000, 75000, 100000, 125000, 150000]
  resources:
    resourceType: "Instances"
    resourceValues: [1, 2, 3, 4, 5]
  slo:
    sloType: "lag trend"
    prometheusUrl: "http://prometheus-operated:9090"
    offset: 0
    properties:
      threshold: 2000
      externalSloUrl: "http://localhost:80/evaluate-slope"
      warmup: 60 # in seconds
  execution:
    strategy: "LinearSearch"
    duration: 300 # in seconds
    repetitions: 1
    loadGenerationDelay: 30 # in seconds
    restrictions:
      - "LowerBound"
  configOVERRIDES: []
```
Theodolite *Executions*

```yaml
apiVersion: theodolite.com/v1
kind: execution
metadata:
  name: theodolite-example-execution
spec:
  benchmark: "uc1-kstreams"
  load:
    loadType: "NumSensors"
    loadValues: [25000, 50000, 75000, 100000, 125000, 150000]
  resources:
    resourceType: "Instances"
    resourceValues: [1, 2, 3, 4, 5]
  slo:
    - sloType: "lag trend"
      prometheusUrl: "http://prometheus-operated:9090"
      offset: 0
      properties:
        threshold: 2000
        externalSloUrl: "http://localhost:80/evaluate-slope"
        warmup: 60 # in seconds
  execution:
    strategy: "LinearSearch"
    duration: 300 # in seconds
    repetitions: 1
    loadGenerationDelay: 30 # in seconds
    restrictions:
      - "LowerBound"
  configOverrrides: []
```
Theodolite *Executions*

Prometheus

SLO evaluation

```yaml
apiVersion: theodolite.com/v1
kind: execution
metadata:
  name: theodolite-example-execution
spec:
  benchmark: "uc1-kstreams"
  load:
    loadType: "NumSensors"
    loadValues: [25000, 50000, 75000, 100000, 125000, 150000]
  resources:
    resourceType: "Instances"
    resourceValues: [1, 2, 3, 4, 5]
  slo:
    sloType: "lag trend"
    prometheusUrl: "http://prometheus-operated:9090"
    offset: 0
    properties:
      threshold: 2000
      externalSloUrl: "http://localhost:80/evaluate-slope"
      warmup: 60 # in seconds
  execution:
    strategy: "LinearSearch"
    duration: 300 # in seconds
    repetitions: 1
    loadGenerationDelay: 30 # in seconds
    restrictions:
      - "LowerBound"
  configOverrides: []
```
### Theodolite Executions

```sh
docker run exec theodolite /opt/theodolite/bin/execlauncher.sh
```

```bash
$ kubectl get executions
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>DURATION</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>osp-flink-execution</td>
<td>PENDING</td>
<td>–</td>
<td>6s</td>
</tr>
<tr>
<td>osp-flink-execution-small</td>
<td>RUNNING</td>
<td>2m</td>
<td>2m52s</td>
</tr>
<tr>
<td>osp-kstreams-default</td>
<td>FINISHED</td>
<td>11h</td>
<td>10d</td>
</tr>
<tr>
<td>osp-kstreams-execution</td>
<td>FINISHED</td>
<td>42m</td>
<td>14d</td>
</tr>
<tr>
<td>osp-spark-streaming-execution</td>
<td>FINISHED</td>
<td>8m</td>
<td>11d</td>
</tr>
<tr>
<td>uc3-flink-template</td>
<td>FINISHED</td>
<td>18m</td>
<td>11d</td>
</tr>
<tr>
<td>uc3-kstreams-default</td>
<td>FINISHED</td>
<td>10h</td>
<td>10d</td>
</tr>
<tr>
<td>uc3-kstreams-default-short</td>
<td>FINISHED</td>
<td>3h</td>
<td>10d</td>
</tr>
<tr>
<td>uc3-kstreams-scotty</td>
<td>FINISHED</td>
<td>4h</td>
<td>10d</td>
</tr>
<tr>
<td>uc3-kstreams-scotty-short</td>
<td>FINISHED</td>
<td>2h</td>
<td>10d</td>
</tr>
</tbody>
</table>
Summary: An Operator for Cloud-native Benchmarks
Summary: An Operator for Cloud-native Benchmarks

clear separation of tool, benchmark and execution
Summary: An Operator for Cloud-native Benchmarks

- clear separation of tool, benchmark and execution
- declarative benchmark description
Summary: An Operator for Cloud-native Benchmarks

- clear separation of tool, benchmark and execution
- declarative benchmark description
- all experimental setup bundled in one file
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark

```java
while {sleep}
```
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark

> `kubectl _`

Utilizing established Kubernetes tools
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark

> kubectl
utilizing established Kubernetes tools

Built-in schema validation by Kubernetes
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark, and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark
- `kubectl` utilization of established Kubernetes tools
- Built-in schema validation by Kubernetes
- Enabling continuous benchmarking with GitOps
Summary: An Operator for Cloud-native Benchmarks

- clear separation of tool, benchmark and execution
- declarative benchmark description
- all experimental setup bundled in one file
- no Kubernetes error handling etc. in benchmark
- `kubectl _`
- utilizing established Kubernetes tools
- built-in schema validation by Kubernetes
- enabling continuous benchmarking with GitOps
- increased implementation effort
Summary: An Operator for Cloud-native Benchmarks

| + | clear separation of tool, benchmark and execution |
| + | declarative benchmark description |
| + | all experimental setup bundled in one file |
| + | no Kubernetes error handling etc. in benchmark |
| + | `kubectl` utilizing established Kubernetes tools |
| + | built-in schema validation by Kubernetes |
| + | enabling continuous benchmarking with GitOps |
| - | increased implementation effort |

However:
Summary: An Operator for Cloud-native Benchmarks

+ clear separation of tool, benchmark and execution
+ declarative benchmark description
+ all experimental setup bundled in one file
+ no Kubernetes error handling etc. in benchmark
+ `kubectl _`
  utilizing established Kubernetes tools
+ [OPENAPI INITIATIVE]
  built-in schema validation by Kubernetes
+ [argo]
  enabling continuous benchmarking with GitOps
+ increased implementation effort

However:
+ new benchmarks may utilize existing operators
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark
- Utilizing established Kubernetes tools
- Built-in schema validation by Kubernetes
- Enabling continuous benchmarking with GitOps
- Increased implementation effort
- New benchmarks may utilize existing operators
- Lots of tooling available for creating new operators

However:
Summary: An Operator for Cloud-native Benchmarks

- Clear separation of tool, benchmark and execution
- Declarative benchmark description
- All experimental setup bundled in one file
- No Kubernetes error handling etc. in benchmark

> kubectl _

Utilizing established Kubernetes tools

- Built-in schema validation by Kubernetes
- Enabling continuous benchmarking with GitOps
- Increased implementation effort

However:

- New benchmarks may utilize existing operators
- Lots of tooling available for creating new operators

https://github.com/cau-se/theodolite