Reliability and Performance Testing for Kubernetes Operators

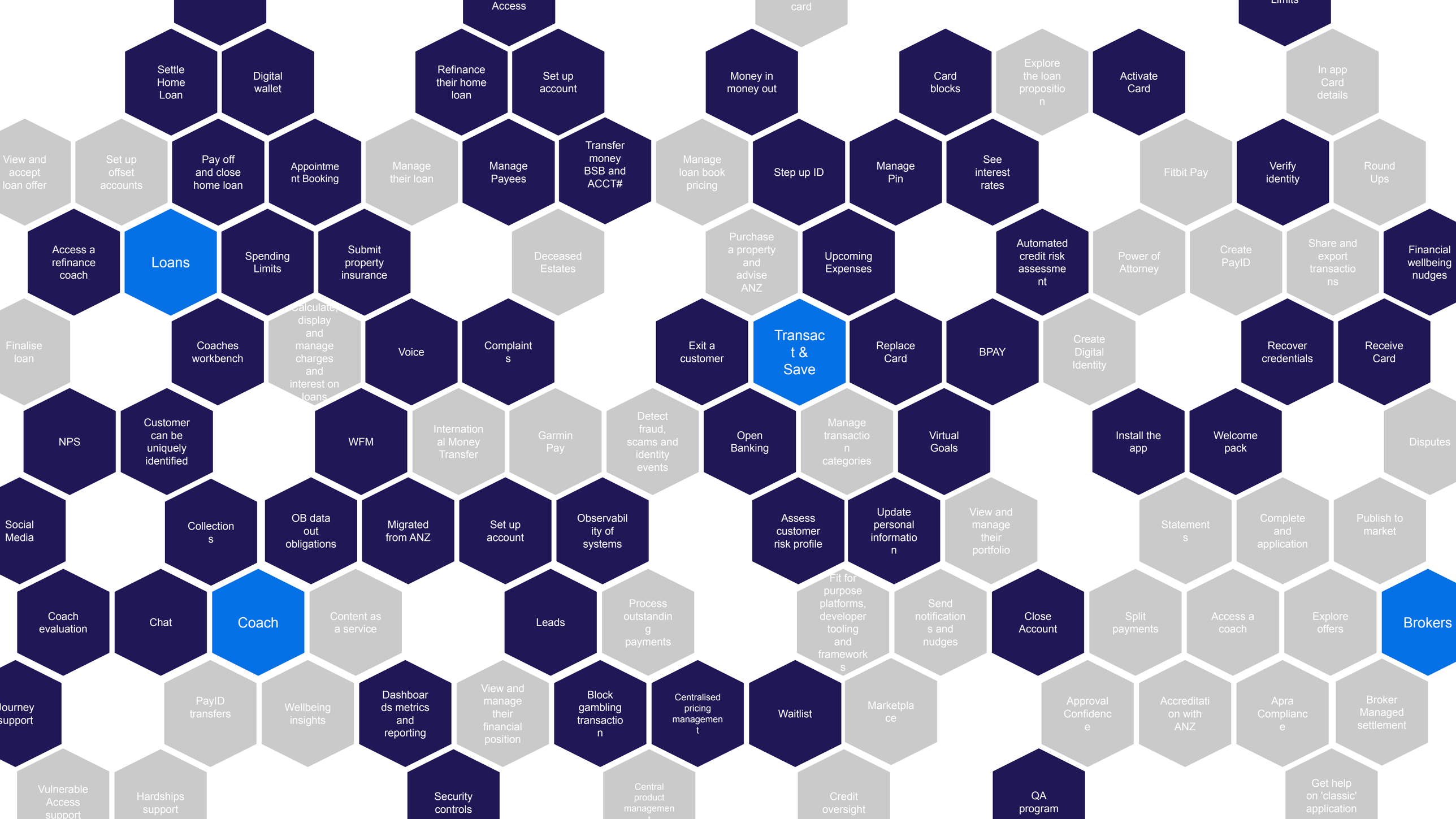
Olga Mirensky

Platform Engineer, ANZ Plus

Agenda

- Introduction
- Why build operators
- Operators performance
- Performance testing
- Testing frameworks
- Q & A

API Driven Platform


Access
card
Limits
Settle Home Loan
Digital wallet
Refinance their home loan
Set up account
Money in money out
Card blocks
Explore the loan propositio n
Activate Card
In app Card details
View and accept loan offer
Set up offset accounts
Pay off and close home loan
Appointme nt Booking
Manage their loan
Manage Payees
Transfer money BSB and ACCT#
Manage loan book pricing
Step up ID
Manage Pin
See interest rates
Fitbit Pay
Verify identity
Round Ups
Access a refinance coach
Loans
Spending Limits
Submit property insurance
Deceased Estates
Purchase a property and advise ANZ
Upcoming Expenses
Automated credit risk assessme nt
Power of Attorney
Create PayID
Share and export transactio ns
Financial wellbeing nudges
Finalise loan
Coaches workbench
Calculate, display and manage charges and interest on loans
Voice
Complaint s
Exit a customer
Transac t & Save
Replace Card
BPAY
Create Digital Identity
Recover credentials
Receive Card
NPS
Customer can be uniquely identified
WFM
Internation al Money Transfer
Garmin Pay
Detect fraud, scams and identity events
Open Banking
Manage transactio n categories
Virtual Goals
Install the app
Welcome pack
Disputes
Social Media
Collection s
OB data out obligations
Migrated from ANZ
Set up account
Observabil ity of systems
Assess customer risk profile
Update personal informatio n
View and manage their portfolio
Statement s
Complete and application
Publish to market
Coach evaluation
Chat
Coach
Content as a service
Leads
Process outstandin g payments
Fit for purpose platforms, developer tooling and framework s
Send notification s and nudges
Close Account
Split payments
Access a coach
Explore offers
Brokers
Journey support
PayID transfers
Wellbeing insights
Dashboar ds metrics and reporting
View and manage their financial position
Block gambling transactio n
Centralised pricing managemen t
Waitlist
Marketpla ce
Approval Confidenc e
Accreditati on with ANZ
Apra Complianc e
Broker Managed settlement
Vulnerable Access support
Hardships support
Security controls
Central product managemen t
Credit oversight
QA program
Get help on 'classic' application

Operators vs Crossplane or False Dichotomy?

Static Client-Side Generation

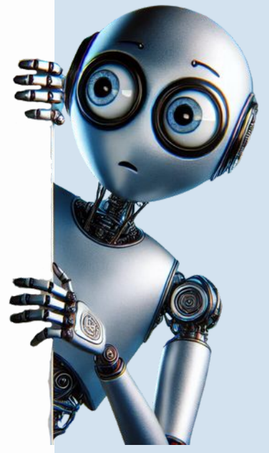
+ Reduce server-side complexity
+ Less runtime resources
+ Predictability
- hard to propagate config or implementation change

Operators

+ Full implementation control
+ Custom business logic and extensibility
+ Event Driven
- Complexity developing and maintaining
- Skillset

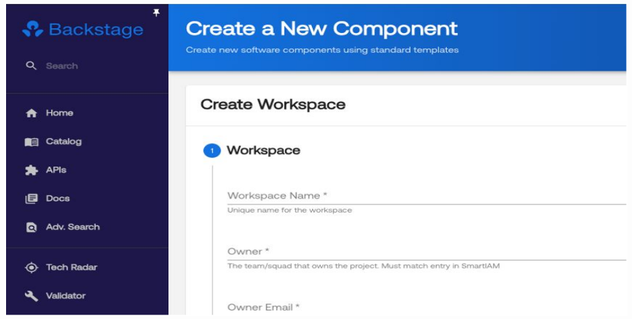
Crossplane / KCC Composition

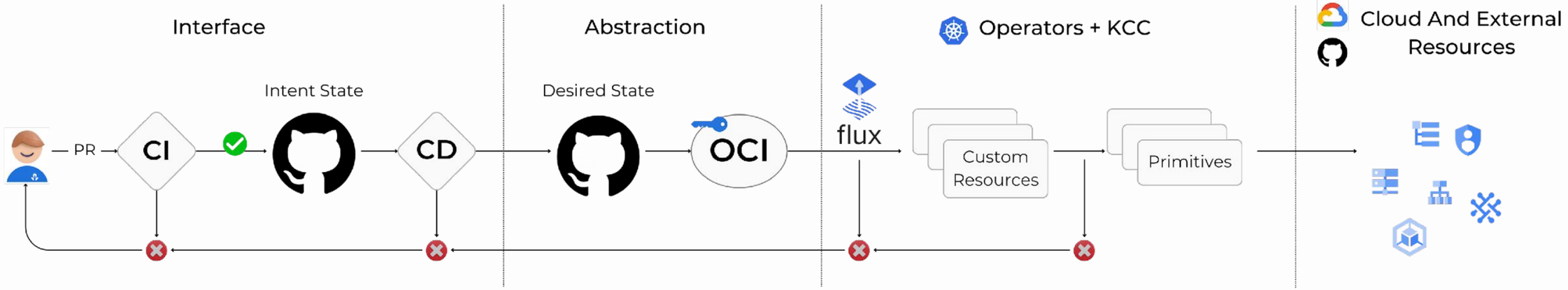
+ No custom code
+ Define high level abstractions with composition
+ Extensibility
- Tool sprawl
- CNCF Incubating project



But why not Both?

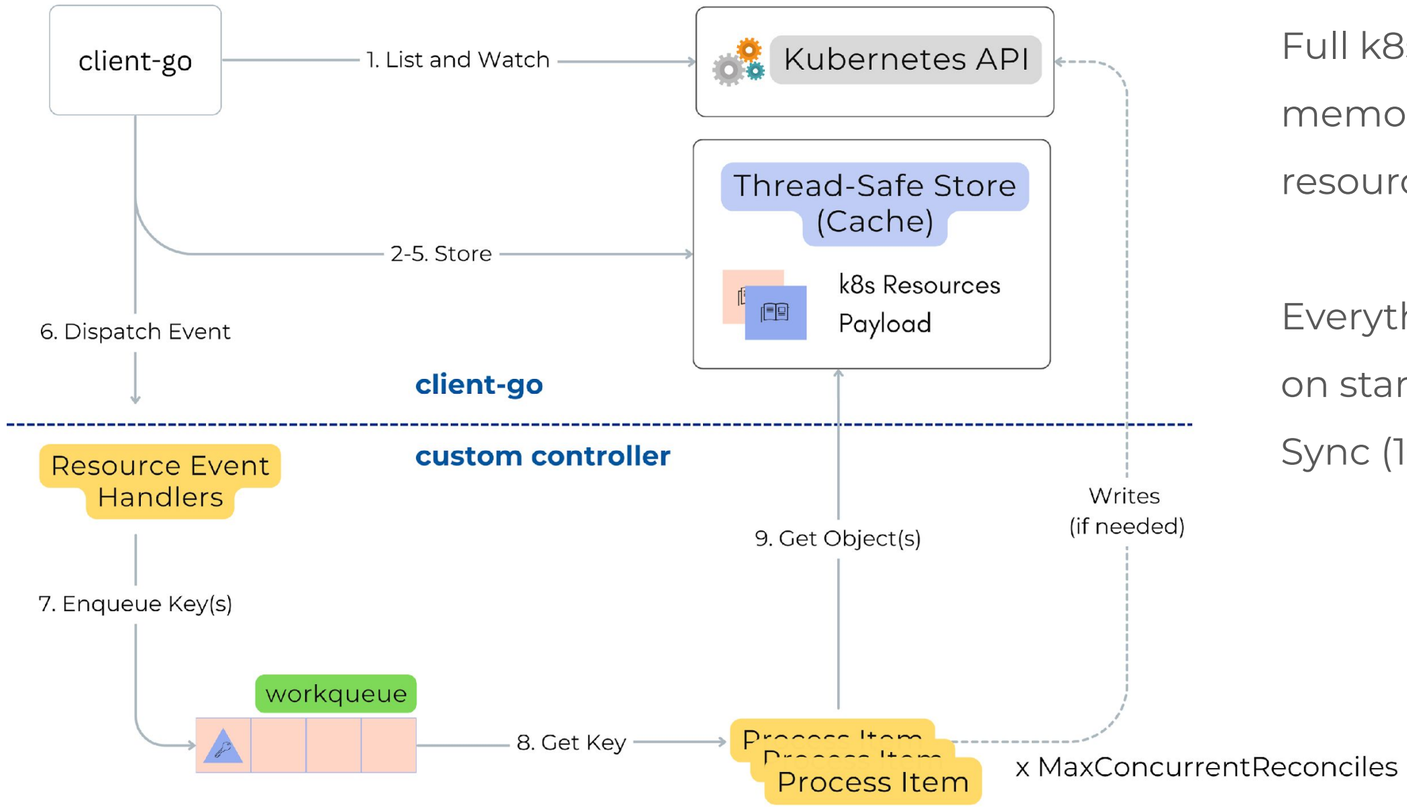
```
$ x workspace create --name demo-workspace --env 'dev, sit, uat, prod' –-enable-kube –-enable-cloudrun
```





Operators Intro and Challenges

Reconciliation Loop

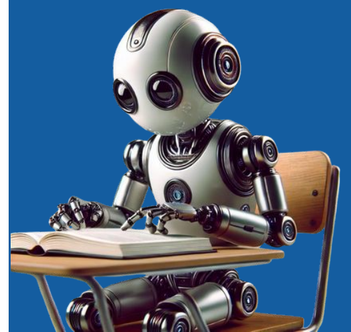


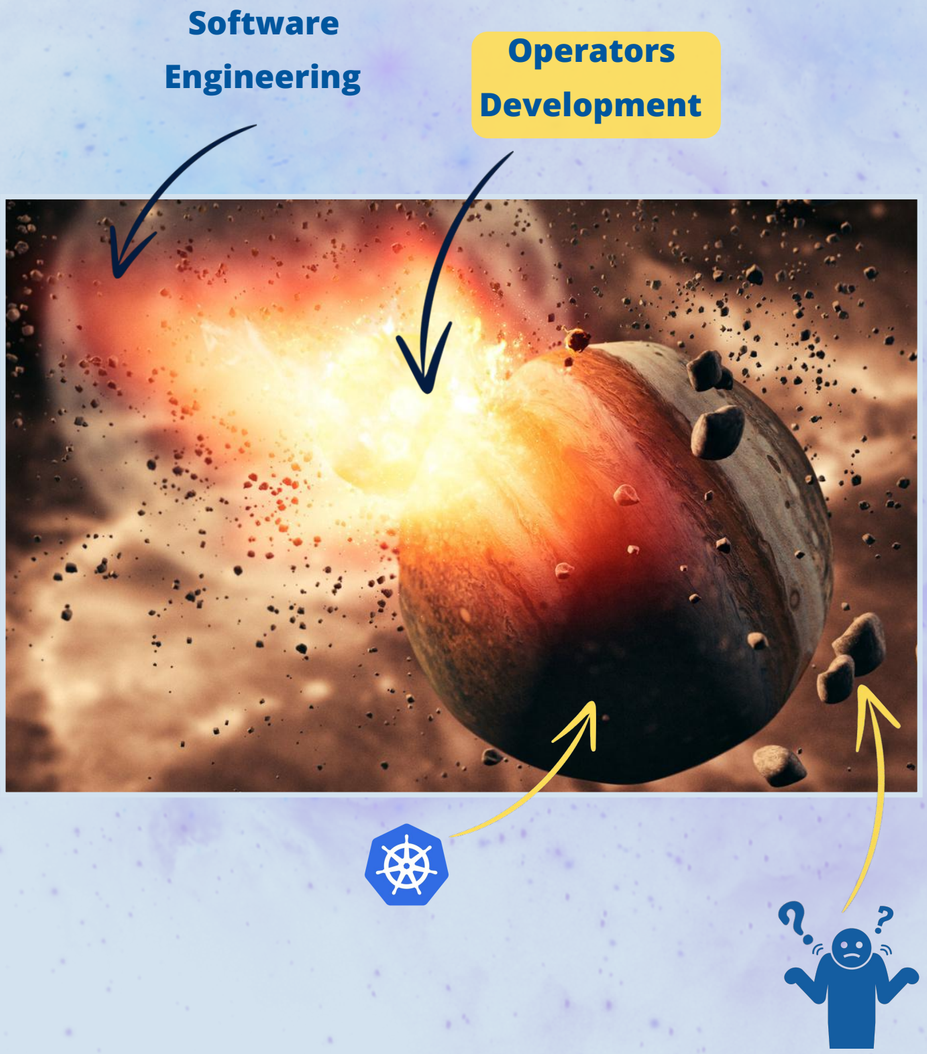
Full k8s payload is stored in memory for all relevant resources for all controllers

Everything is reconciled at once on start-up and each Cache Sync (10hrs)

Based on ”Client-go under the hood”

https://github.com/kubernetes/sample-controller/blob/master/docs/controller-client-go.md





Operators: Software Engineering meets Kubernetes

Unit Test — Isolated Go functions and logic.

Envtest — Simulated in-memory k8s API server

e2e Test — Declarative test in a real cluster (cloud or kind)

Performance Testing ← **Today's focus**

Performance Testing

Performance Testing

Load Test

Test how system performs under load close to normal expected levels. Benchmark system performance to catch regressions

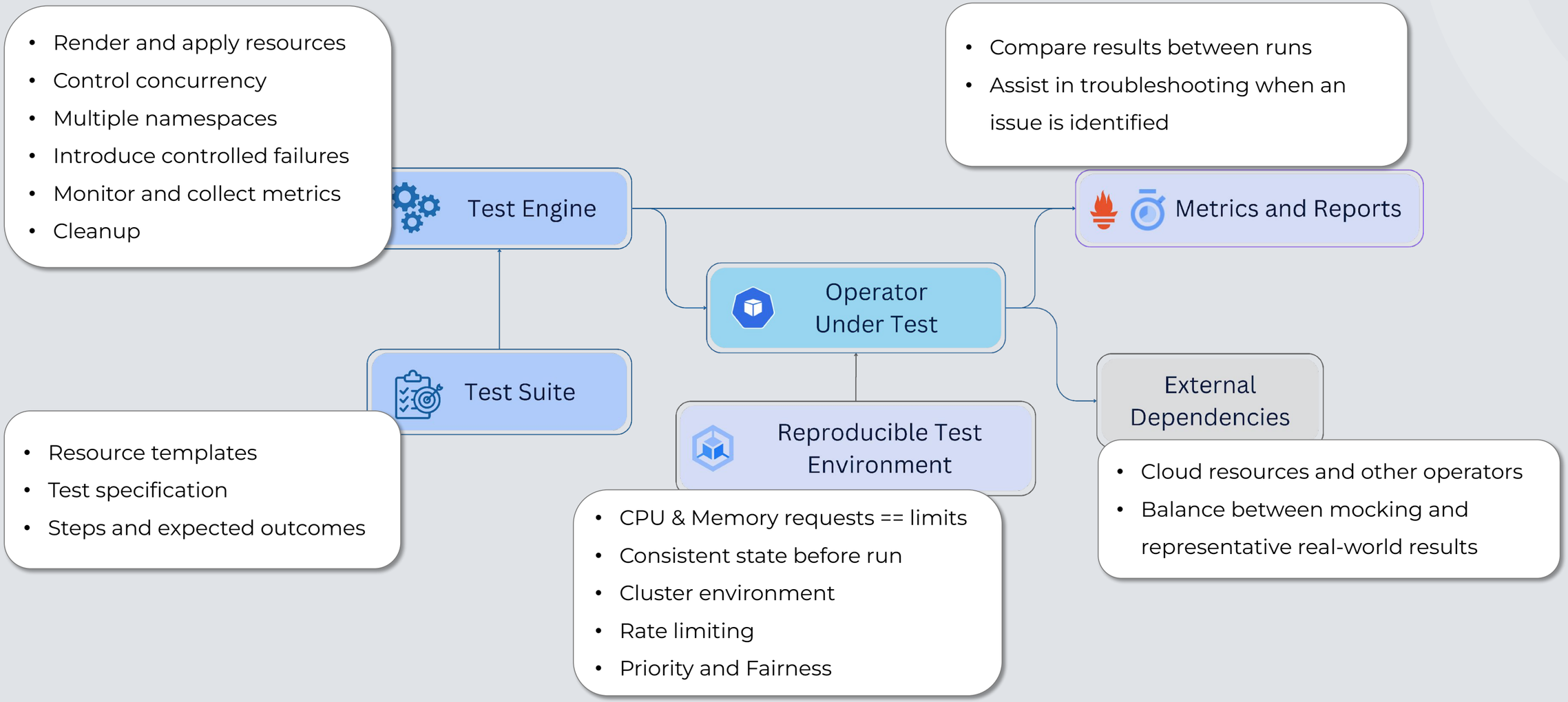
Stress Test

Test for system limits and breaking points by applying load higher than anticipated in normal operation. Useful for capacity planning, test graceful failure, plan Disaster Recovery.

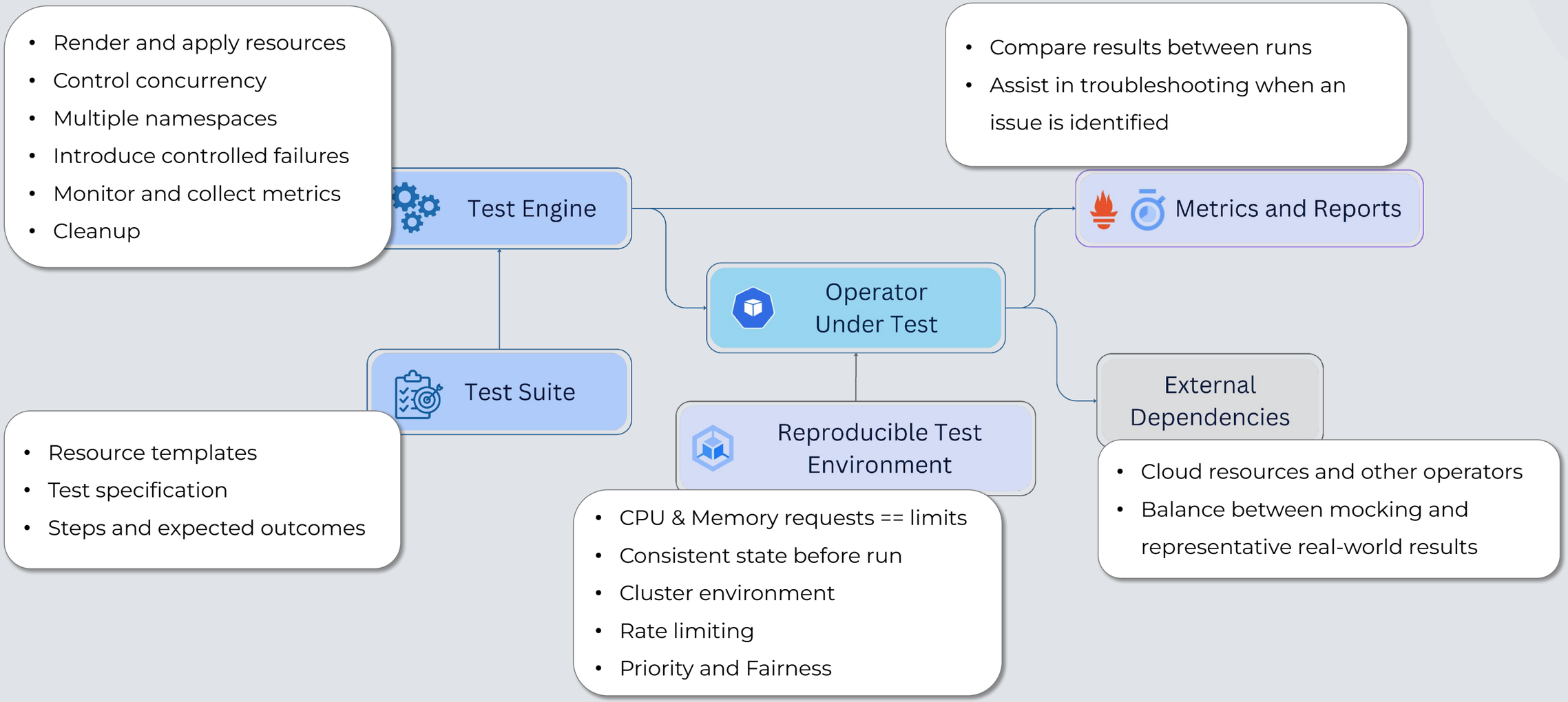
Objectives

- Define SLOs
- Capacity planning
- Regression detection
- Performance troubleshooting
- Inform design decisions

Load Test


Render and apply resources
Control concurrency
Multiple namespaces
Introduce controlled failures
Monitor and collect metrics
Cleanup
Test Engine
Compare results between runs
Assist in troubleshooting when an issue is identified
Metrics and Reports
Operator Under Test
Test Suite
Resource templates
Test specification
Steps and expected outcomes
Reproducible Test Environment
CPU & Memory requests == limits
Consistent state before run
Cluster environment
Rate limiting
Priority and Fairness
External Dependencies
Cloud resources and other operators
Balance between mocking and representative real-world results

Load Test


Render and apply resources
Control concurrency
Multiple namespaces
Introduce controlled failures
Monitor and collect metrics
Cleanup
Test Engine
Compare results between runs
Assist in troubleshooting when an issue is identified
Metrics and Reports
Operator Under Test
Test Suite
Resource templates
Test specification
Steps and expected outcomes
Reproducible Test Environment
CPU & Memory requests == limits
Consistent state before run
Cluster environment
Rate limiting
Priority and Fairness
External Dependencies
Cloud resources and other operators
Balance between mocking and representative real-world results

Operators Load Test Challenges

01

No Standard Load Testing Framework

No generic, standardised framework specifically designed for load testing Kubernetes operators.

02

CRs != HTTP Requests

Standard load testing tools are built with web traffic in mind. Can be used if operator’s functionality is exposed as API.

03

Targets and Dimensions

Unlike web-based apps load testing which are usually defined in terms of end-user volume, for operators it is harder to identify targets – number of services, teams or custom attributes?

04

External Resources

Creating external resources may incur undesirable cost. Emulating dependencies on the other hand, may lead to results that are not representative of real production performance.

Testing Frameworks

Declarative e2e Testing

```yaml
apiVersion: chainsaw.kyverno.io/v1alpha1
kind: Test
metadata:
  name: example
spec:
  concurrent: false
  timeouts:
    apply: 10s
    assert: 10s
    error: 10s
  steps:
  - try:
    - apply:
        file: ../resources/good-resource.yaml
  - try:
    - assert:
        file: ../resources/expected-result.yaml
  - try:
    - error:
        file: ../resources/bad-resource.yaml
```

```
=== NAME  chainsaw/single-cnp
    l.go:53: | 14:33:41 | single-cnp | step-01  | PATCH    | OK    | netoperator.platform.x.anz/
    l.go:53: | 14:33:41 | single-cnp | step-01  | APPLY    | DONE  | netoperator.platform.x.anz/
    l.go:53: | 14:33:41 | single-cnp | step-01  | ASSERT   | RUN   | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:41 | single-cnp | step-01  | ASSERT   | DONE  | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:41 | single-cnp | step-01  | TRY      | DONE  |
    l.go:53: | 14:33:41 | single-cnp | step-02  | TRY      | RUN   |
    l.go:53: | 14:33:41 | single-cnp | step-02  | APPLY    | RUN   | netoperator.platform.x.anz/
    l.go:53: | 14:33:41 | single-cnp | step-02  | PATCH    | OK    | netoperator.platform.x.anz/
    l.go:53: | 14:33:41 | single-cnp | step-02  | APPLY    | DONE  | netoperator.platform.x.anz/
    l.go:53: | 14:33:41 | single-cnp | step-02  | ASSERT   | RUN   | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:41 | single-cnp | step-02  | ASSERT   | DONE  | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:41 | single-cnp | step-02  | TRY      | DONE  |
    l.go:53: | 14:33:41 | single-cnp | step-00  | CLEANUP  | RUN   |
    l.go:53: | 14:33:41 | single-cnp | step-00  | DELETE   | RUN   | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:42 | single-cnp | step-00  | DELETE   | OK    | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:42 | single-cnp | step-00  | DELETE   | DONE  | networking.k8s.io/v1/Networ
    l.go:53: | 14:33:42 | single-cnp | step-00  | DELETE   | RUN   | v1/Namespace @ test-cluster
    l.go:53: | 14:33:42 | single-cnp | step-00  | DELETE   | OK    | v1/Namespace @ test-cluster
```

Kyverno

Cluster Loader 2

https://github.com/kubernetes/perf-tests/blob/master/clusterloader2/docs/design.md

+ Powerful test engine
+ Templating support
+ Concurrency and QPS support
+ Support for Generic Custom Resources
+ Step types: run, measure, report.
+ Built-in Prometheus or BYO
+ Open Source
+ Chaos features – need to explore more.
+ We don’t need to write our own tool

👏

- Not designed as a generic operator testing framework
- Initial learning curve – docs are not descriptive enough, requires trial and error and diving in source code.
- OOTB measurements aimed for testing k8s components.
- Built in prometheus is not straight-forward.

test.yaml

```
name: LoadTest-MyTest

tuningSets:
  - name: Parallel5
    qpsLoad:
      qps: 10
      parallelism: 4

steps:
- name: run test
  phases:
    - tuningSet: Parallel5
      replicasPerNamespace: 50
      namespaceRange:
        min: 1
        max: 20
      objectBundle:
        - basename: cl2-cr
          objectTemplatePath: ”my-cr-tmpl.yaml”
          templateFillMap:
            varName: “foo”
```

Concurrency configurable per phase

Deploy resources across namespaces “base-<sha>-1” to “base-<sha>-20”

Template vars

my-cr-tmpl.yaml

```
apiVersion: example.com/v1alpha1
kind: MyCr
metadata:
  labels:
    app: label
  name: {{.Name}}
  namespace: {{.Namespace}}
spec:
  myKey: {{.varName}}
  restOfTheSpec: ....
```

Measurements and Metrics

```
- name: Wait for objects to be ready
  measurements:
    - Method: WaitForGenericK8sObjects
      Identifier: WaitForMyCR
      Params:
        objectGroup/Version/Resource
        namespaceRange: ...
        timeout: 200s
        successfulConditions:
        - Ready=True
        minDesiredObjectCount: 1
        maxFailedObjectCount: 1
```

More methods supported for core k8s components metrics

```
- name: Start measurements
  measurements:
    - Identifier: gq
      Method: GenericPrometheusQuery
      Params:
        action: start
        metricName: Controller reconcile
        metricVersion: v1
        unit: total
        queries:
          - name: reconcileTimeSeconds
            query: <prom-query>

- name: Gather measurements
  measurements:
    - Identifier: gq
      Method: GenericPrometheusQuery
      Params:
        action: gather
        enableViolations: true
```

Q & A