



Developing Operators with
Kubernetes Operator
Pythonic Framework
KOPF



KubeCon San Diego
2019-11-20



SERGEY VASILYEV
[@nolar](https://twitter.com/nolar)



About me

- My name is **Sergey Vasilyev**
 - <https://twitter.com/nolar>
- A Python developer from Siberia. Now in Berlin, Germany.
- ~20 years of experience in Software Engineering.
- In love with Python since 2009.
- Sr. Backend Engineer, Zalando SE.
- Running ML apps & infra on Kubernetes for Zalando Pricing & Forecasting.

The image shows the Zalando logo, which consists of a large orange teardrop shape on the left and the word "Zalando" in a bold, black, sans-serif font on the right. The logo is mounted on a light gray wall. The text "ABOUT US" is overlaid in white, bold, sans-serif font across the center of the logo.

ABOUT US

ZALANDO AT A GLANCE

~ **5.4** billion EUR
revenue 2018

> **300**
million visits
per
month

> **450,000**
product choices

~ **14,000**
employees in
Europe

> **80%**
of visits via
mobile devices

> **29**
million
active customers

> **2,000**
brands

17
countries

WE ARE CONSTANTLY INNOVATING TECHNOLOGY

**HOME-BREWED,
CUTTING-EDGE
& SCALABLE**
technology solutions



help our brand to
WIN ONLINE



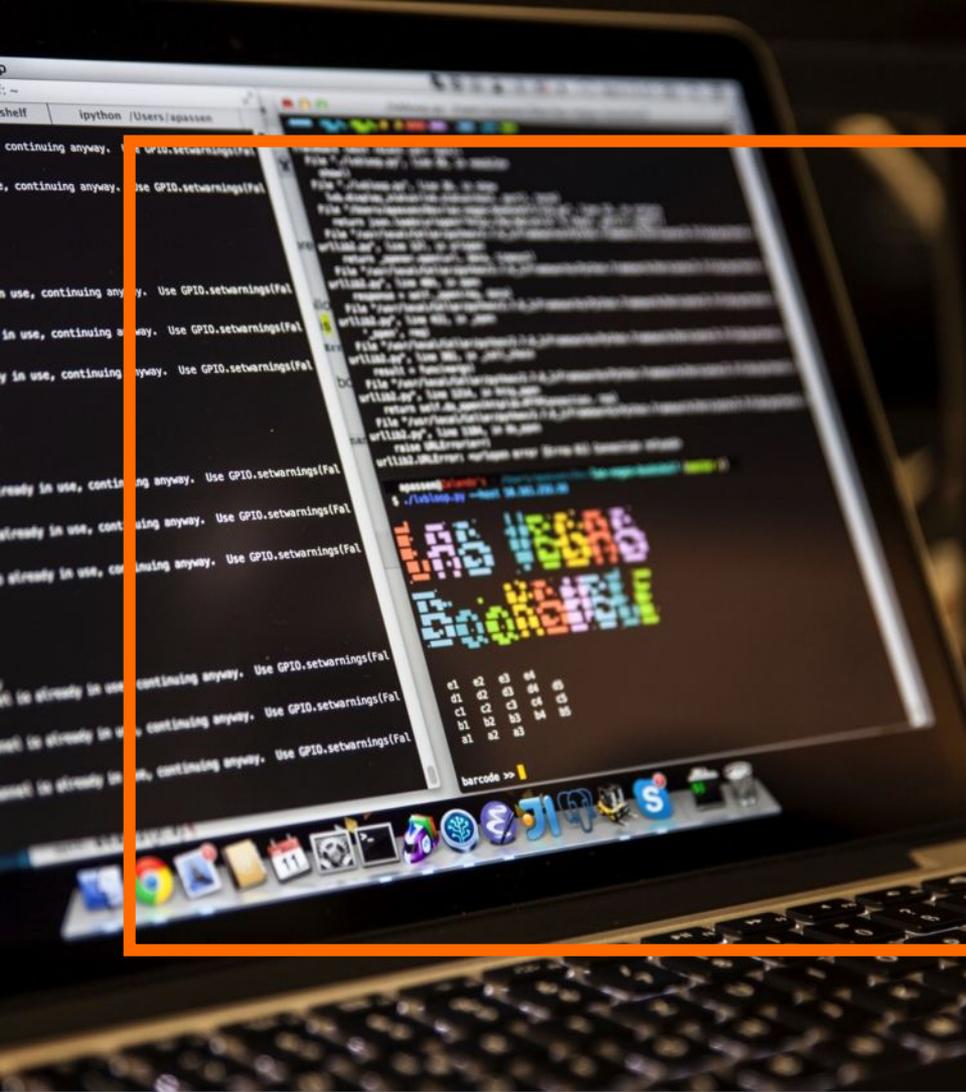
7 international
tech locations



> 2,000
employees at

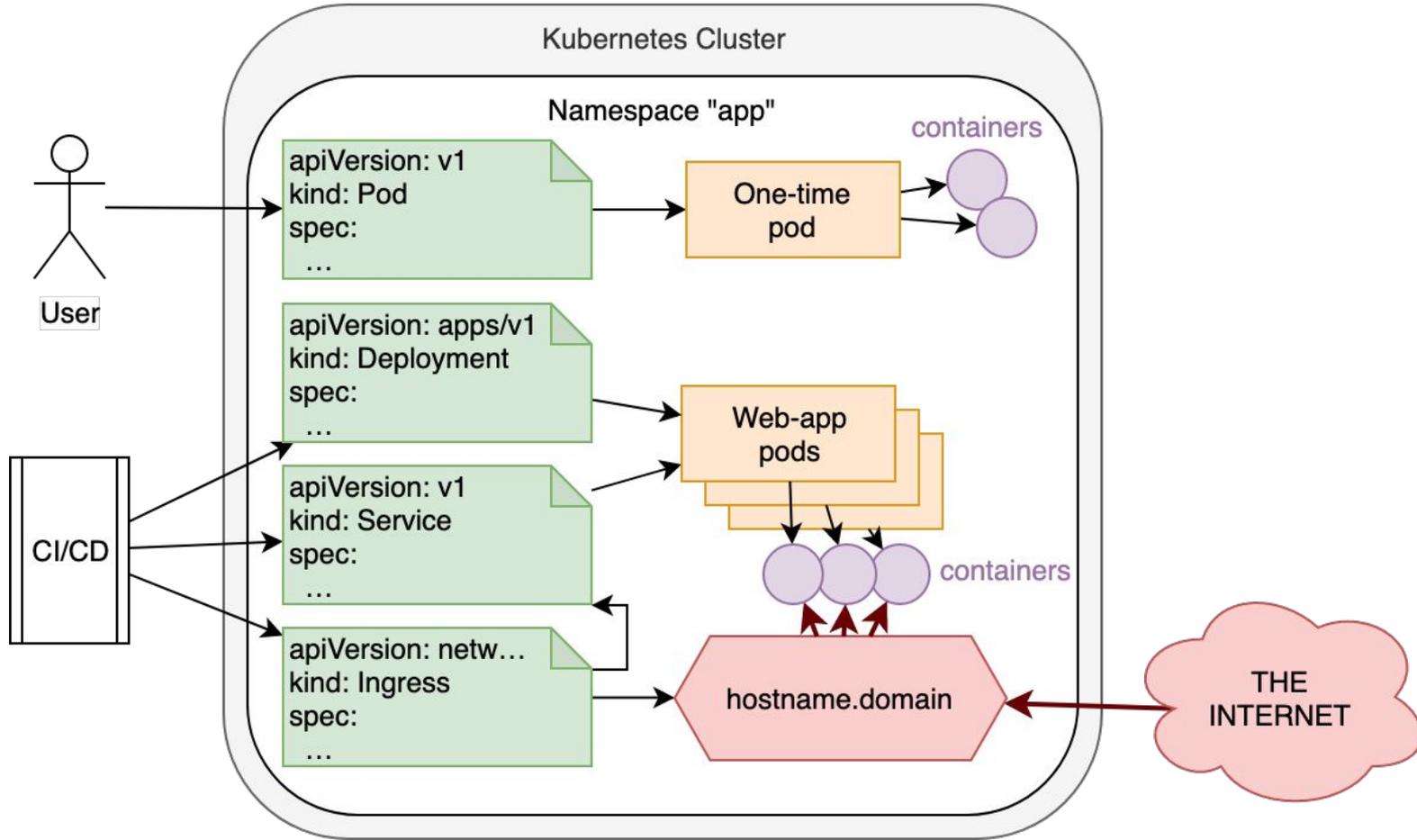
HQs
in Berlin



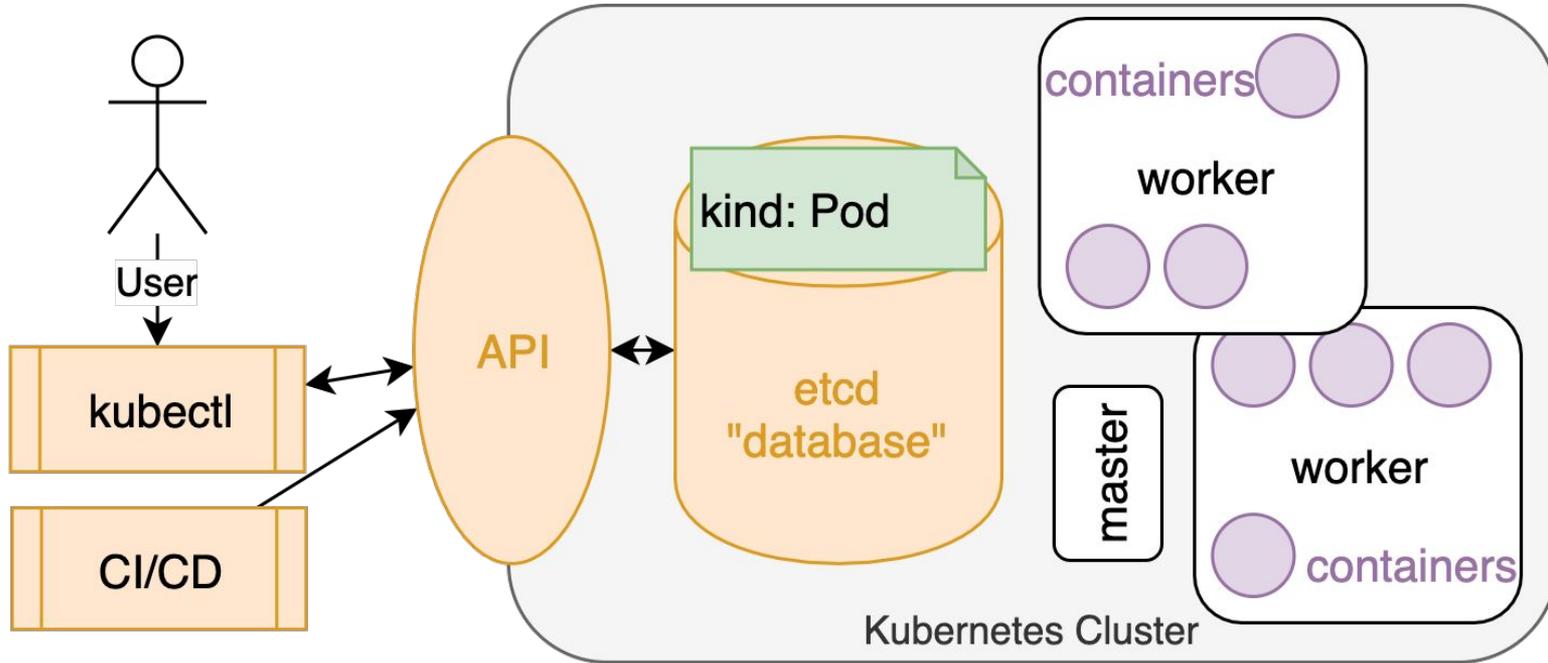


WE DRESS CODE

“Kubernetes is a container orchestrator”



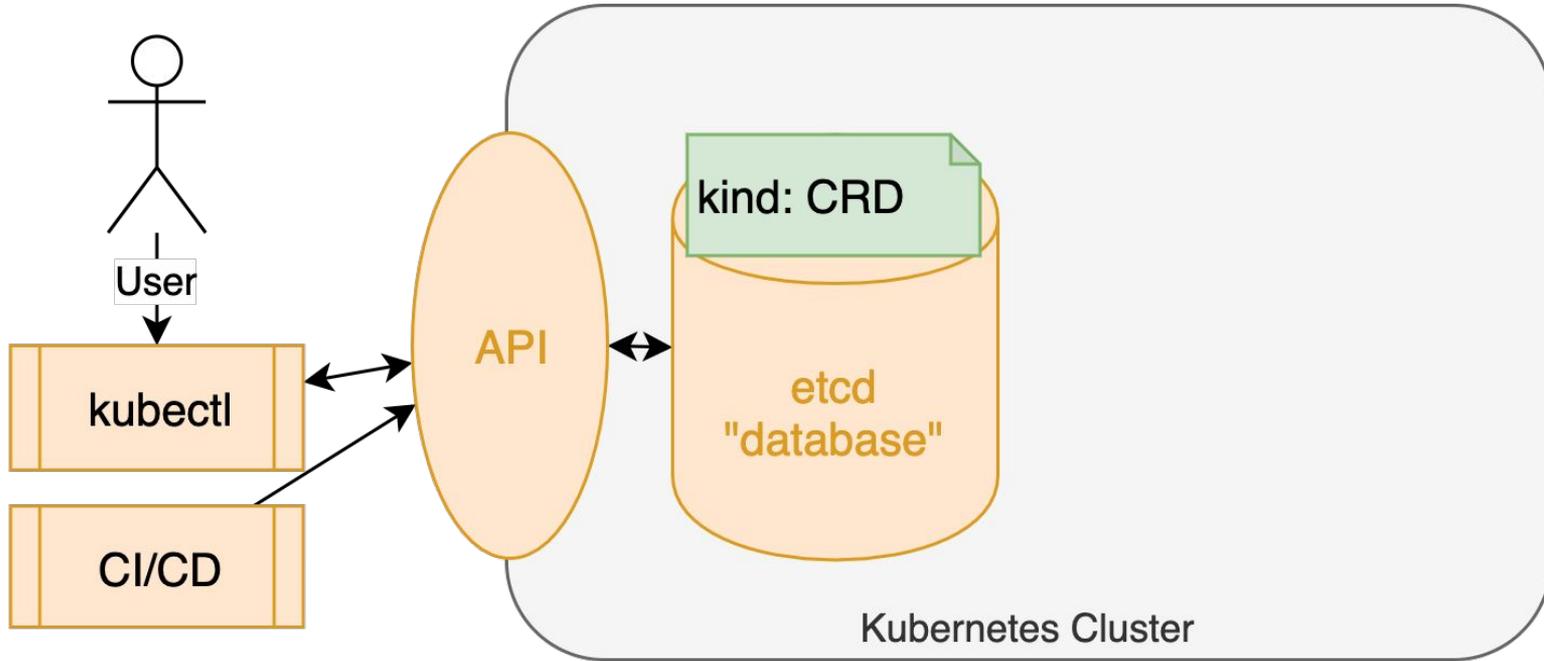
Kubernetes under the hood



```
kubectl get pods
kubectl create pod ...
kubectl patch pod ...
kubectl delete pod ...
```

```
GET /api/v1/pods
POST ...
PATCH ...
DELETE ...
```

Extending Kubernetes: Custom Resource Definitions



```
kubectl get kopfexamples
kubectl create kex ...
kubectl patch kex ...
kubectl delete kex ...
```

```
GET /apis/zalando.org/v1/namespaces/default/kopfexamples
POST ...
PATCH ...
DELETE ...
```

Defining a custom resource

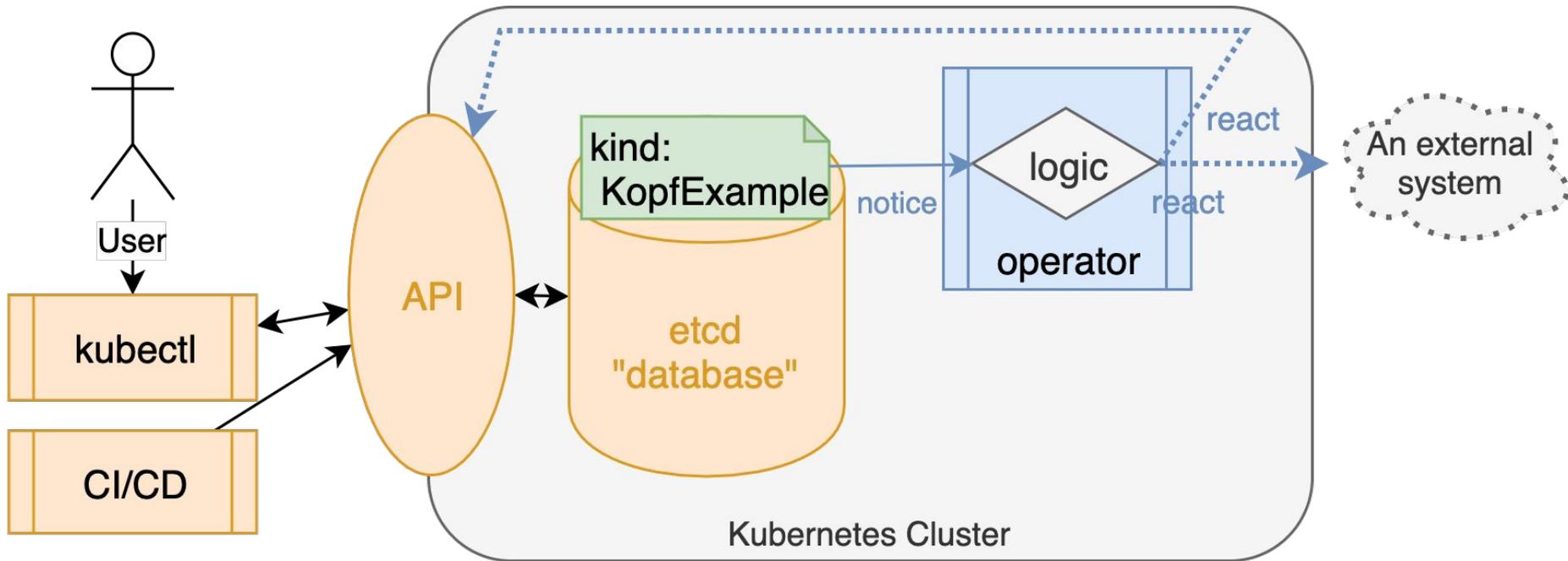
- **Required:** “group/version”.
- **Required:** kind/plural/singular names.
- **Required:** scope (“Namespaced”).
- **Optional:** short names (aliases).
- **Optional:** list formatting and columns.
- **Optional:** categories.

```
apiVersion: zalando.org/v1
kind: KopfExample
metadata:
  name: kopf-example-1
  labels:
    somelabel: somevalue
  annotations:
    someannotation: somevalue
spec:
  duration: 1m
  field: value
  items:
  - item1
  - item2
```

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: kopfexamples.zalando.org
spec:
  scope: Namespaced
  group: zalando.org
  versions:
  - name: v1
    served: true
    storage: true
  names:
    kind: KopfExample
    plural: kopfexamples
    singular: kopfexample
    shortNames:
    - kopfexes
    - kopfex
    - kex
```

```
$ kubectl apply -f crd.yaml
$ kubectl apply -f obj.yaml
```

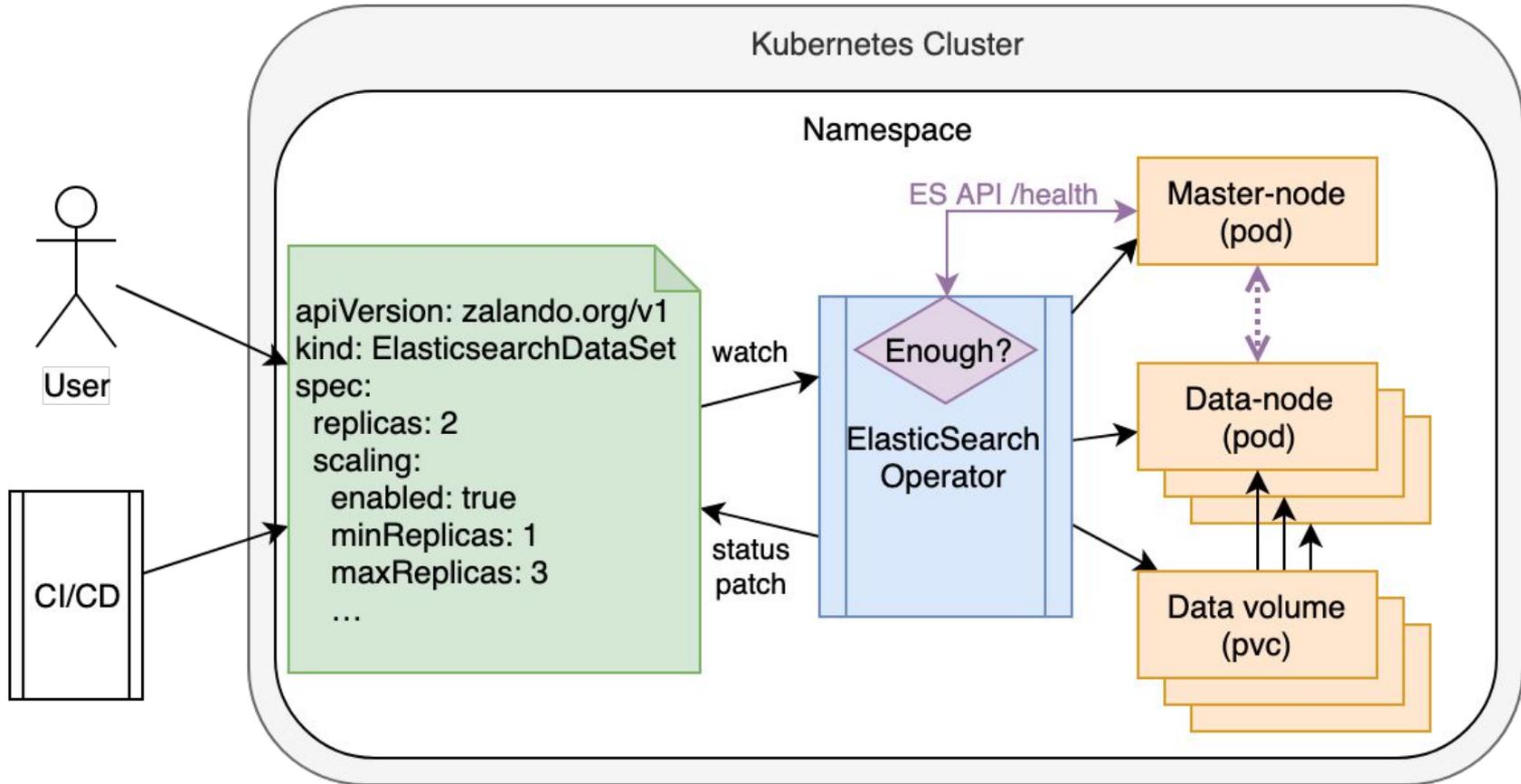
Extending Kubernetes: Controllers/Operators



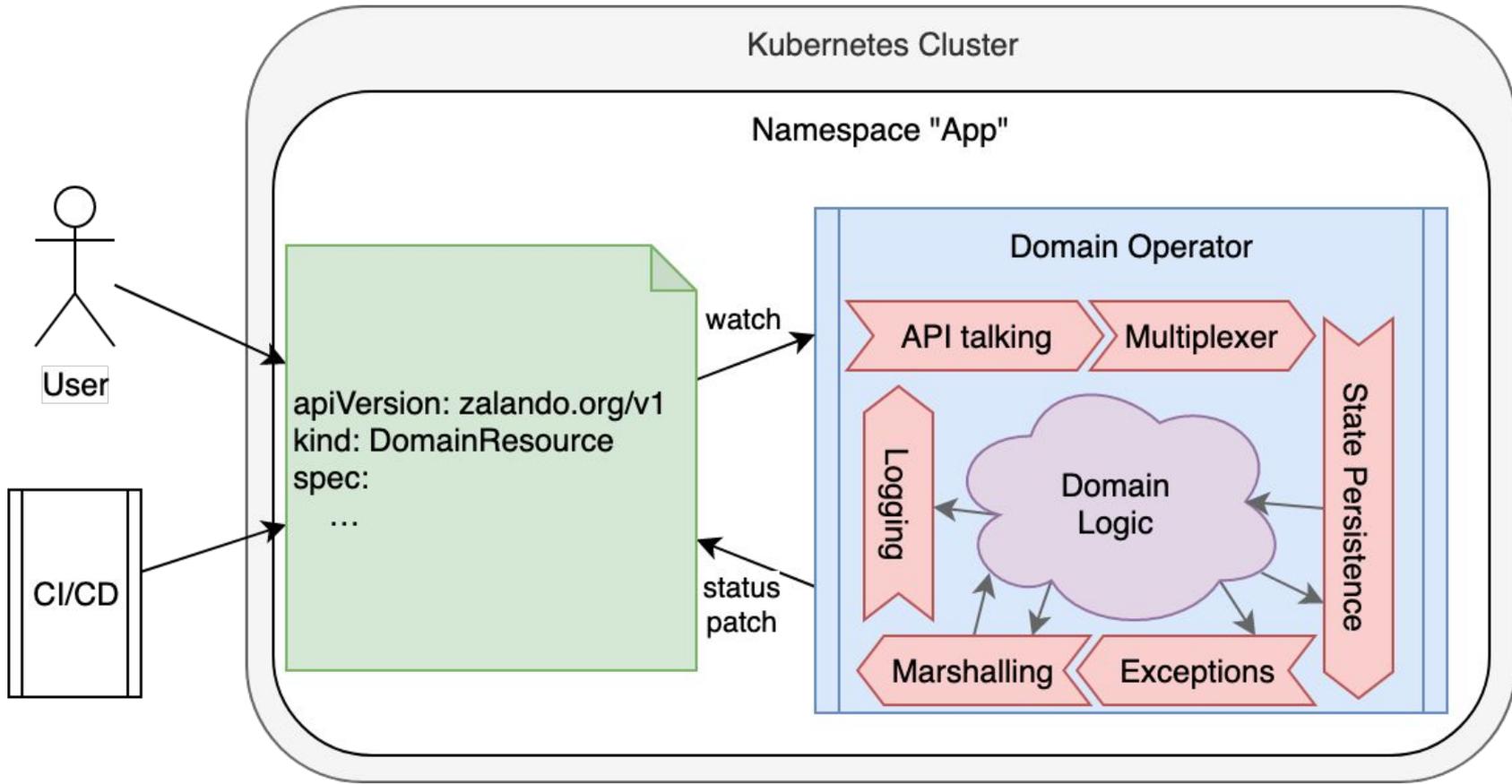
```
kubectl get kopfexamples  
kubectl create kex ...  
kubectl patch kex ...  
kubectl delete kex ...
```

```
GET /apis/zalando.org/v1/namespaces/default/kopfexamples  
POST ...  
PATCH ...  
DELETE ...
```

Common use: an application-specific operator



Common problem: infrastructure code hassle





MAKE A FRAMEWORK!



WILLKOMMEN, KOPF!

The simplest Kubernetes operator

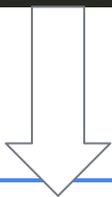
```
import kopf

@kopf.on.create('zalando.org', 'v1', 'kopfexamples')
def create_fn(spec, logger, **kwargs):
    logger.info("%s", f"And here we go: {spec}")
```

The simplest Kubernetes operator

```
1 import kopf
2
3
4 @kopf.on.create('zalando.org', 'v1', 'kopfexamples')
5 def create_fn(spec, **kwargs):
6     print(f"And here we are! Creating: {spec}")
7     return {'message': 'hello world'} # will be the new status
```

```
1 # A demo custom resource for
2 apiVersion: zalando.org/v1
3 kind: KopfExample
4 metadata:
5   name: kopf-example-1
6   labels:
7     somelabel: somevalue
8 spec:
9   duration: 1m
10  field: value
11  items:
12  - item1
13  - item2
```



```
$ kopf run scripts.py [--verbose]
```

```
And here we are! Creating: {'duration': '1m', 'field': 'value',
'items': ['item1', 'item2']}
```

```
[2019-02-25 14:06:54,742] kopf.reactor.handler [INFO   ]
[default/kopf-example-1] Handler create_fn succeeded.
```

```
[2019-02-25 14:06:54,856] kopf.reactor.handler [INFO   ]
[default/kopf-example-1] All handlers succeeded for creation.
```

```
$ kubectl apply -f obj.yaml
$ kubectl describe -f obj.yaml
```

```
Name:          kopf-example-1
```

```
...
```

```
Status:
```

```
  create_fn:
```

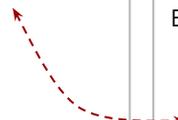
```
    Message: hello world
```

```
Events:
```

Type	Reason	Age	From	Message
------	--------	-----	------	---------

----	-----	----	----	-----
------	-------	------	------	-------

Normal	Success	81s	kopf	Handler create_fn succeeded.
--------	---------	-----	------	------------------------------



Resource-watching handlers

- As often, as the events arrive from K8s API.
- Raw data, no interpretation.
- Fire-and-forget, errors are ignored.
- Similar to `kubectl get --watch`

```
1 import kopf
2
3
4 @kopf.on.event('zalando.org', 'v1', 'kopfexamples')
5 def event_fn_with_error(**kwargs):
6     raise Exception("Oops!")
7
8
9 @kopf.on.event('zalando.org', 'v1', 'kopfexamples')
10 def normal_event_fn(event, **kwargs):
11     print(f"Event received: {event!r}")
12
```

Change-detection handlers

```
1 import kopf
2
3 @kopf.on.create('zalando.org', 'v1', 'kopfexamples')
4 def create_fn_1(spec, **kwargs):
5     print(f'CREATED 1st: field={spec.field}')
6
7 @kopf.on.create('zalando.org', 'v1', 'kopfexamples')
8 def create_fn_2(meta, **kwargs):
9     print(f'CREATED 2nd: name={meta["name"]}')
10
11 @kopf.on.update('zalando.org', 'v1', 'kopfexamples')
12 def update_fn(old, new, diff, **kwargs):
13     print(f'UPDATED: diff={diff}')
14
15 @kopf.on.delete('zalando.org', 'v1', 'kopfexamples')
16 def delete_fn_1(**kwargs):
17     print('DELETED')
18
19 @kopf.on.field('zalando.org', 'v1', 'kopfexamples', field='spec.field')
20 def field_fn(old, new, **kwargs):
21     print(f'FIELD CHANGED: {old} -> {new}')
```

Error handling

```
1 import kopf
2
3 @kopf.on.create('zalando.org', 'v1', 'kopfexamples')
4 def instant_failure_with_only_a_message(**kwargs):
5     raise kopf.PermanentError("Fail once and for all.")
6
7 @kopf.on.create('zalando.org', 'v1', 'kopfexamples')
8 def eventual_success_with_few_messages(retry, **kwargs):
9     if retry < 3: # 0, 1, 2, 3
10        raise kopf.TemporaryError("Expected recoverable error.", delay=1.0)
11
12 @kopf.on.create('zalando.org', 'v1', 'kopfexamples', retries=3)
13 def eventual_failure_with_tracebacks(**kwargs):
14     raise Exception("An error that is supposed to be recoverable.")
15
16 @kopf.on.create('zalando.org', 'v1', 'kopfexamples', errors=kopf.ErrorsMode.PERMANENT)
17 def instant_failure_with_traceback(**kwargs):
18     raise Exception("An error that is supposed to be recoverable.")
```

Debugging & breakpoints

```
1 import kopf
2
3 @kopf.on.create('zalando.org',
4 def instant_failure_with_only_a
5     raise kopf.PermanentError("
6
7 @kopf.on.create('zalando.org',
8 def eventual_success_with_few_m
9     if retry < 3: # 0, 1, 2, 3
10         raise kopf.TemporaryErr
11
12 @kopf.on.create('zalando.org',
13 def eventual_failure_with_trace
14     raise Exception("An error t
15
16 @kopf.on.create('zalando.org', 'v1', 'kopfexamples', errors=kopf.ErrorsMode.PER
17 def instant_failure_with_traceback(**kwargs):
18     raise Exception("An error that is supposed to be recoverable.")
```

The screenshot shows a Python debugger window titled "Debug: example". The interface is split into three main sections: "Frames", "Variables", and "Console".

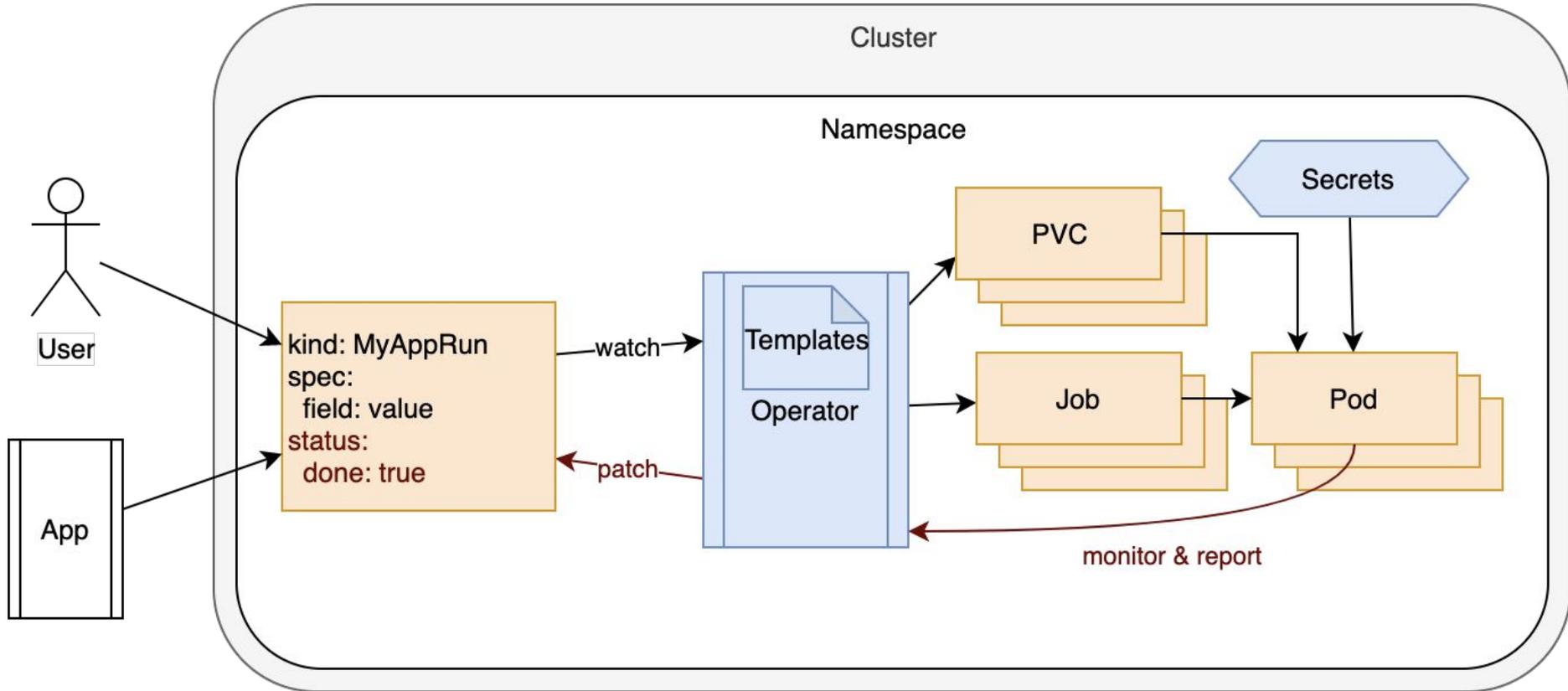
- Frames:** Shows a stack trace with the following frames from top to bottom:
 - ThreadP...utor-0_5 (selected)
 - eventual_failure_with_tracebacks, (selected)
 - run, thread.py:57
 - _worker, thread.py:80
 - run, threading.py:865
 - _bootstrap_inner, threading.py:9
 - _bootstrap, threading.py:885
- Variables:** Shows the state of variables in the current frame:
 - kwargs = {dict} ... Loading Value
 - retry (4322537400) = {int} 0
 - started (4301378368) = {datetime} 2019-11-
 - runtime (4348040112) = {timedelta} 0:00:00.
 - cause (4303316728) = {ResourceChangingCa
 - logger (4308588672) = {ObjectLogger} <Obj
 - patch (4346603984) = {Patch} {}
 - memo (4303759880) = {ObjectDict} {}
 - body (4303124384) = {dict} <class 'dict': { 'a
 - spec (4299098352) = {DictView} {'duration':
 - meta (4300346456) = {DictView} {'annotatio
 - status (4301378592) = {DictView} {'kopf': { 'p
 - uid (4308588280) = {str} '07572bed-1810-4e
 - name (4298958080) = {str} 'kopf-example-1'
 - namespace (4299918640) = {str} 'default'
 - event (4303802808) = {Reason} create
 - reason (4298997240) = {Reason} create
 - diff (4317499944) = {Diff} (('add', ()), None, {
 - old (4299029616) = {NoneType} None
 - new (4299029560) = {dict} <class 'dict': { 's
- Console:** Currently empty.

Other things

- Python logging → Kubernetes Events.
- Custom authentication → @on.login
- @on.startup / @on.cleanup
- Embeddable.
- Testing toolkit.
- Resource hierarchies.
- *And more...*

- *Other patterns? [Create a feature request!](#)*

Cross-resource relations



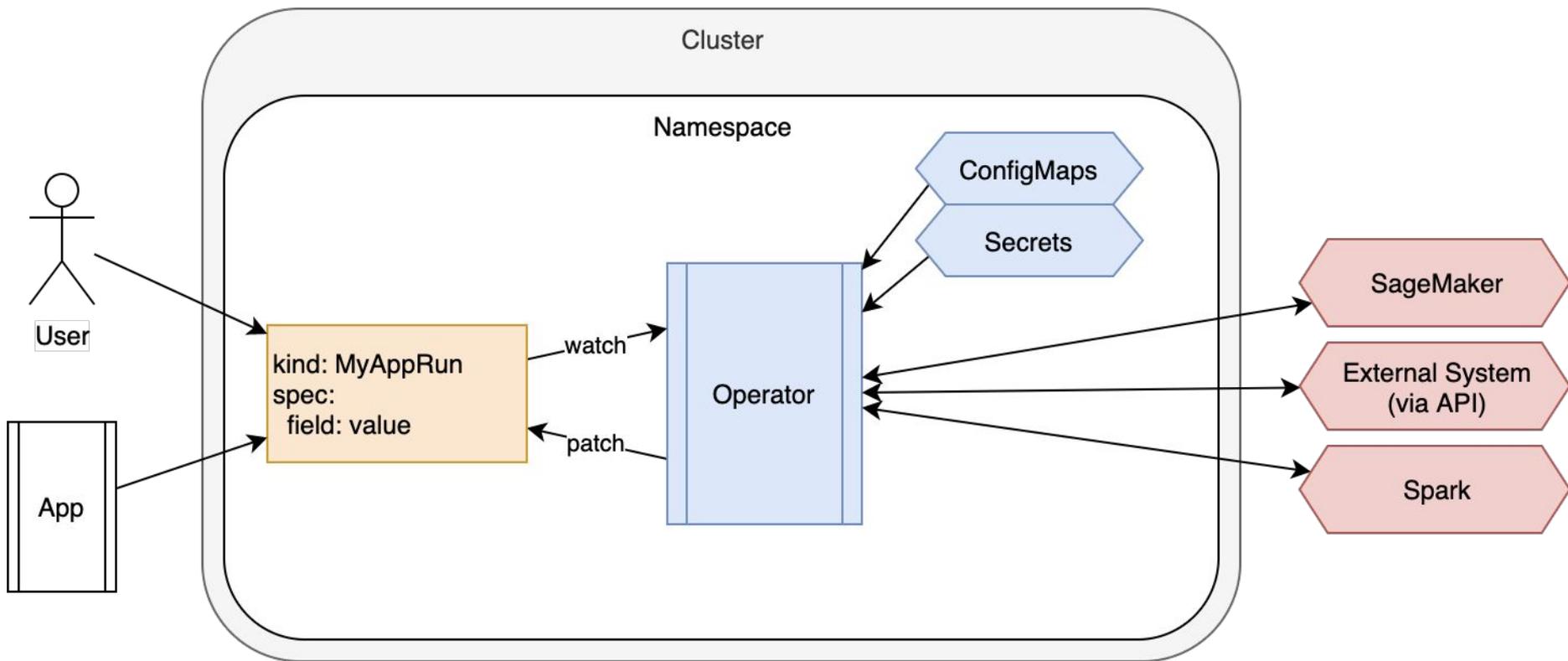
Cross-resource relations

```
@kopf.on.create('zalando.org', 'v1', 'kopfexamples')
def kex_created(spec, name, **kwargs):
    doc = yaml.safe_load("""...""")
    kopf.adopt(doc)
    kopf.label(doc, {'parent-name': name}, nested=['spec.template'])

    child = pykube.Job(api, doc)
    child.create()
```

```
@kopf.on.event('', 'v1', 'pods', labels={'parent-name': None})
def event_in_a_pod(meta, status, namespace, **kwargs):
    phase = status.get('phase') # make a decision!
    query = KopfExample.objects(api, namespace=namespace)
    parent = query.get_by_name(meta['labels']['parent-name'])
    parent.patch({'status': {'children': phase}})
```

Kubernetes is an orchestrator of everything



Kubernetes is an orchestrator of everything

```
@kopf.on.create('zalando.org', 'v1', 'kopfexamples')
@kopf.on.resume('zalando.org', 'v1', 'kopfexamples')
async def created(body, logger, memo, name, namespace, **kwargs):
    task = asyncio.create_task(bg(name, namespace, logger))
    memo.task = task

@kopf.on.delete('zalando.org', 'v1', 'kopfexamples')
async def deleted(memo, **kwargs):
    if hasattr(memo, 'task'):
        memo.task.cancel()
        try:
            await memo.task
        except Exception:
            pass

async def bg(name, namespace, logger):
    while True:
        try:
            logger.info("Still alive. Checking...")
            some_remote_progress = random.randint(0, 100)

            api = pykube.HTTPClient(pykube.KubeConfig.from_env())
            obj = KopfExample.objects(api, namespace=namespace).get_by_name(name)
            obj.patch({'status': {'message': f'{some_remote_progress}% done'}})

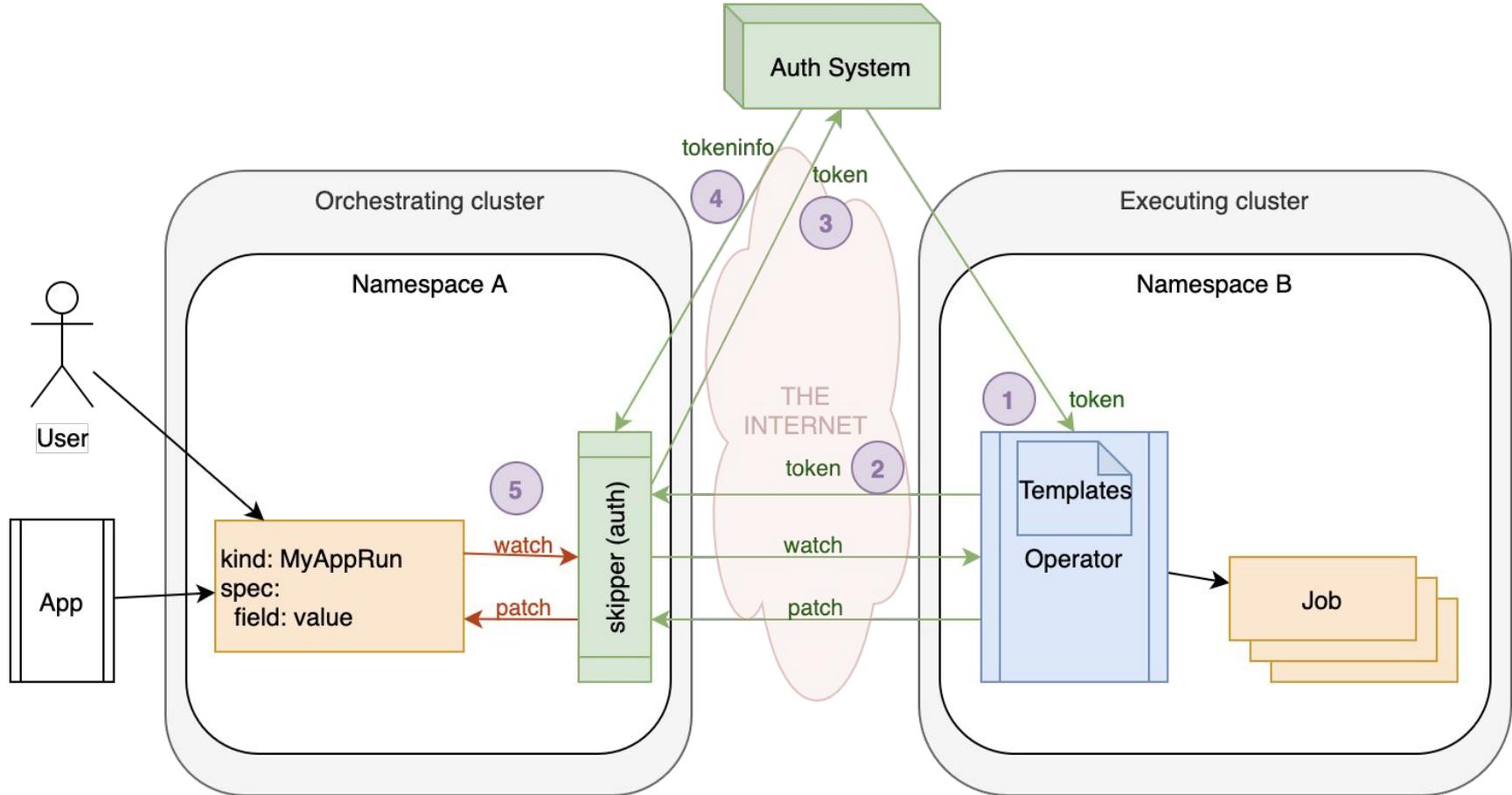
        except Exception as e:
            logger.error("Dead: %r", e)
        else:
            await asyncio.sleep(10)
```

Kubernetes is an orchestrator of everything

Coming
soon!

```
@kopf.on.timer('zalando.org', 'v1', 'kopfexamples', interval=10)
async def check_it(logger, patch, **kwargs):
    logger.info("Checking...")
    some_remote_progress = random.randint(0, 100)
    patch.setdefault('status', {})['message'] = f'{some_remote_progress}%'
```

Cross-cluster connectivity



Cross-cluster connectivity

```
import kopf

@kopf.on.login()
def remote_cluster_a(**kwargs):
    return kopf.ConnectionInfo(
        server='https://remote-kubernetes/',
        token='abcdef123')

@kopf.on.create('zalando.org', 'v1', 'kopfexamples')
def kex_created(name, **kwargs):
    print(name)
```

FUTURE

- More patterns: cross-resource handlers.
- Reconciliation handlers: by time; threads/tasks.
- Admission handlers: validation/mutation.

- SDK: YAML from Python — CRDs, RBAC, Deployments.
- Operator Lifecycle Manager integration.
- More Kopf-based operators.

- Bring Python to Kubernetes, build a community.
- Conquer the world [of Kubernetes operators].

SUMMARY

- Kubernetes operators can be easy.
- Kubernetes operators can be ad-hoc.
- Kubernetes operators can be Pythonic.
- Kubernetes is an orchestrator of everything.
- Focus on the domain, not on the infrastructure.
- Simplicity & human-friendliness as the #1 priority.
- Python community as a huge leverage for Kubernetes.
- Use Kopf.

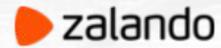
LINKS

- Source: <https://github.com/zalando-incubator/kopf/>
- Examples: <https://github.com/zalando-incubator/kopf/tree/master/examples>
- Documentation: <https://kopf.readthedocs.io/>

- Me (Sergey Vasilyev):
 - Twitter: [@nolar](https://twitter.com/nolar)
- Us (Zalando SE):
 - Twitter: [@ZalandoTech](https://twitter.com/ZalandoTech) & <https://jobs.zalando.com/tech/blog/>

- Other Zalando operators (not Kopf-based):
 - ElasticSearch operator: <https://github.com/zalando-incubator/es-operator>
 - Postgres operator: <https://github.com/zalando/postgres-operator>
- Worth reading:
 - Kubernetes Failure Stories: <https://k8s.af/>





QUESTIONS?



SERGEY VASILYEV
SENIOR BACKEND ENGINEER



sergey.vasilyev@zalando.de

[@nolar](https://www.instagram.com/nolar)



Why Python? — Scale x15!

