Build Your Kubernetes Operator With the Right Tool!

...

Rafal Leszko
@RafalLeszko
rafalleszko.com

About me

- Distributed Software Engineer at Livepeer
- Worked at Hazelcast, Google, and CERN
- Author of the book "Continuous Delivery with Docker and Jenkins"
- Trainer and conference speaker
- Live in Kraków, Poland

Agenda

- Introduction
- Tools for building Operators
  - Operator SDK
    - Helm
    - Ansible
    - Go
  - Operator Frameworks: KOPF, Java Operator SDK, ...
  - Bare Programming Language: Java, Kotlin, C#, ...
- Summary

Introduction

Kubernetes Operator

Kubernetes Operator

Operator is an application that watches a custom Kubernetes resource and performs some operations upon its changes.

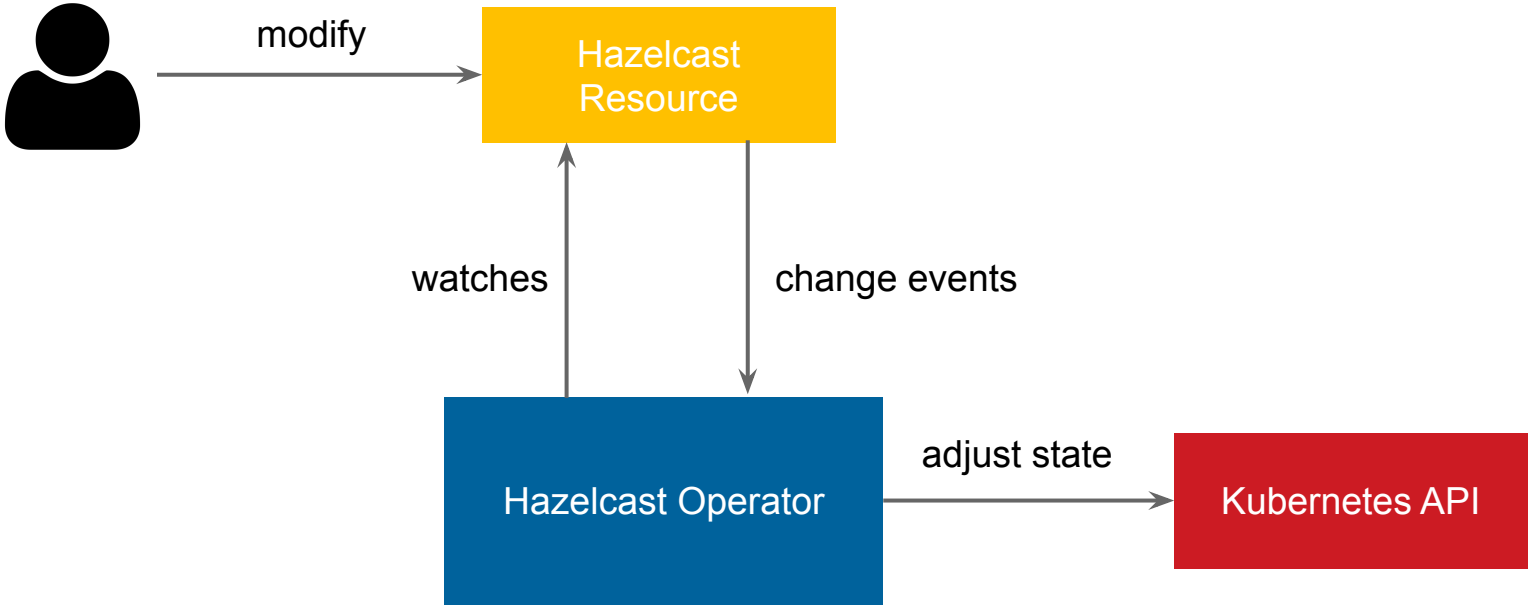
Resource (hazelcast.yaml)

```yaml
apiVersion: hazelcast.my.domain/v1
kind: Hazelcast
metadata:
  name: hazelcast-sample
spec:
   size: 1
```

Resource (hazelcast.yaml)

```yaml
apiVersion: hazelcast.my.domain/v1
kind: Hazelcast
metadata:
  name: hazelcast-sample
spec:
   size: 1
```

```
$ kubectl apply -f hazelcast.yaml
```


modify
Hazelcast Resource
watches
change events
Hazelcast Operator
adjust state
Kubernetes API

OPERATOR
SDK

Kubernetes Operators

FRAMEWORKS

KOPF

Operator Similarities

Step 1: Implement the operator logic

Operator Similarities

Step 1: Implement the operator logic

Step 2: Dockerize your operator

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
```

Operator Similarities

Step 1: Implement the operator logic

Step 2: Dockerize your operator

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
```

Step 3: Create CRD (Custom Resource Definition)

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

Operator Similarities

Operator Similarities

Step 4: Create RBAC (Role and Role Binding)

```
$ kubectl apply -f role.yaml
$ kubectl apply -f role_binding.yaml
```

Operator Similarities

Step 4: Create RBAC (Role and Role Binding)

```
$ kubectl apply -f role.yaml
$ kubectl apply -f role_binding.yaml
```

Step 5: Deploy the operator

```
$ kubectl apply -f operator.yaml
```

Operator Similarities

Step 4: Create RBAC (Role and Role Binding)

```
$ kubectl apply -f role.yaml
$ kubectl apply -f role_binding.yaml
```

Step 5: Deploy the operator

```
$ kubectl apply -f operator.yaml
```

Step 6: Create your custom resource

```
$ kubectl apply -f hazelcast.yaml
```

Operator Similarities

Step 1: Implement the operator logic

Step 2: Dockerize your operator

Step 3: Create CRD (Custom Resource Definition)

Step 4: Create RBAC (Role and Role Binding)

Step 5: Deploy the operator

Step 6: Create your custom resource

Agenda

- Introduction ✔
- Tools for building Operators
  - Operator SDK
    - Helm
    - Ansible
    - Go
  - Operator Frameworks: KOPF, Java Operator SDK, ...
  - Bare Programming Language: Java, Kotlin, C#, ...
- Summary

Operator SDK: Helm

Helm is a package manager for Kubernetes.

It allows you to:

- create templated Kubernetes configurations
- package them into a Helm chart
- render them using parameters defined in values.yaml

Helm: Create Helm Chart

```
$ helm create chart
```

```
# chart/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "hazelcast.fullname" . }}
spec:
  replicas: {{ .Values.size }}
  selector:
    matchLabels:
      app: hazelcast
```

```
# chart/values.yaml
size: 1
```

```
template:
  metadata:
    labels:
      app: hazelcast
  spec:
    containers:
      - name: hazelcast
        image: "hazelcast/hazelcast"
```

Helm: Generate Operator

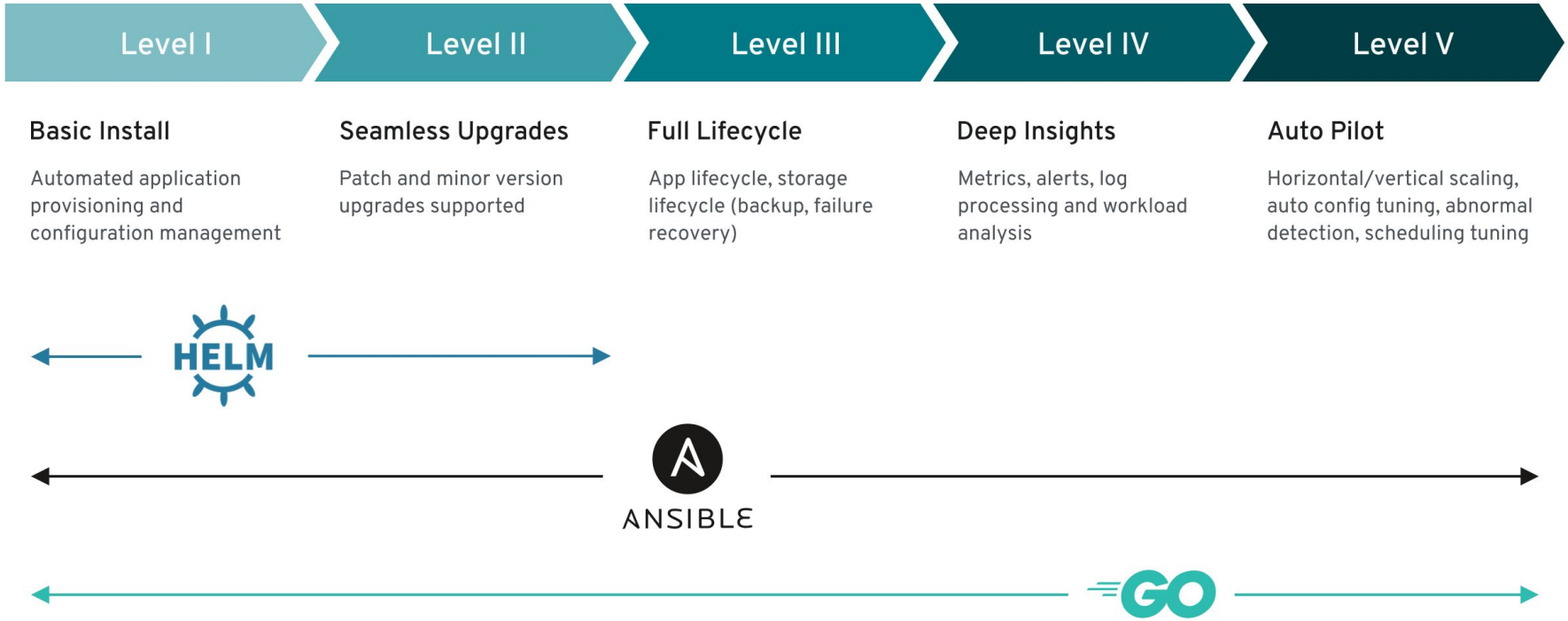
Generate Hazelcast Operator

```
$ operator-sdk init --plugins=helm
$ operator-sdk create api --group=hazelcast --version=v1
--helm-chart=./chart
```

Common Operator Steps: build, install, and use

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
$ make install
$ make deploy IMG=<user>/hazelcast-operator
$ kubectl apply -f config/samples/hazelcast_v1_hazelcast.yaml
```

Operator SDK: Operator Capability Levels



What does "Operator SDK: Helm" mean to You?

- Implementation is declarative and simple
- If you already have a Helm chart, then no work at all
- Limited to the features available in Helm
- Operator manifest/configuration files are automatically generated

Operator SDK: Ansible

Ansible is a very powerful tool for IT automation.

It allows you to:

- create tasks in a declarative way
- perform complex DevOps tasks using simple YAML files
- interact with Kubernetes API using the community.kubernetes.k8s plugin

Ansible: Create Operator

```
$ operator-sdk init --plugins=ansible
$ operator-sdk create api --group hazelcast --version v1 --kind Hazelcast \
--generate-role

# roles/hazelcast/tasks/main.yml
---
- name: start hazelcast
  community.kubernetes.k8s:
    definition:
      kind: Deployment
      apiVersion: apps/v1
      metadata:
        name: hazelcast
        namespace: '{{ansible_operator_meta.namespace}}'
      spec:
        replicas: "{{size}}"
```

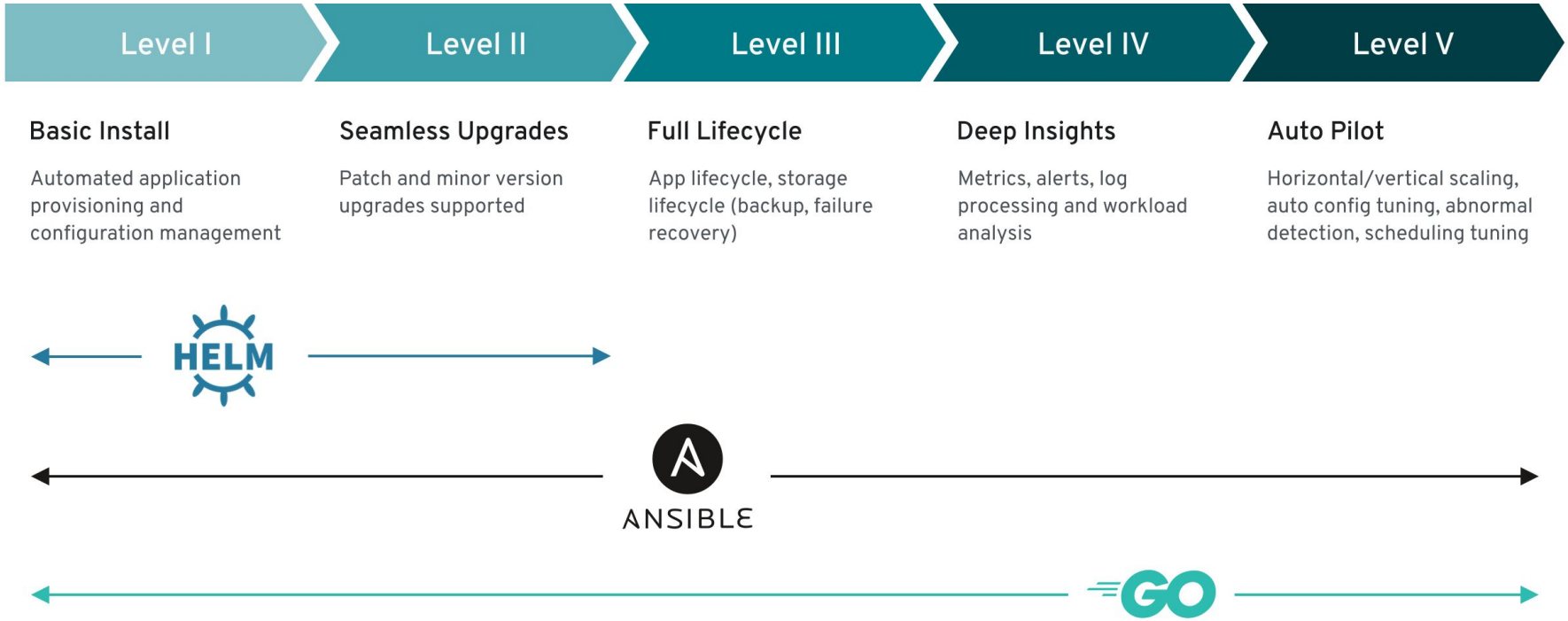
```
selector:
  matchLabels:
    app: hazelcast
template:
  metadata:
    labels:
      app: hazelcast
  spec:
    containers:
    - name: hazelcast
      image: "hazelcast/hazelcast"
```

Ansible: Build, Install, Use Operator

Common Operator Steps: build, install, and use

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
$ make install
$ make deploy IMG=<user>/hazelcast-operator
$ kubectl apply -f config/samples/hazelcast_v1_hazelcast.yaml
```

Operator SDK: Operator Capability Levels



What does "Operator SDK: Ansible" mean to You?

- Implementation is declarative and therefore concise and human-readable
- YAML configuration is executed via the community.kubernetes.k8s plugin
- Ansible covers all capability levels
- Operator manifest/configuration files are automatically generated

Operator SDK: Go

Go is a general-purpose programming language.

Advantages:

- Kubernetes is written in Go
- good Kubernetes client library
- performance

Go: Create Operator

```
$ operator-sdk init --repo=github.com/<user>/hazelcast-operator
$ operator-sdk create api --version v1 --group=hazelcast --kind Hazelcast --resource=true
--controller=true
```

```
// controllers/hazelcast_controller.go
// +kubebuilder:rbac:groups=hazelcast.my.domain,resources=hazelcasts,verbs=get;list…
func (r *HazelcastReconciler) Reconcile(req ctrl.Request) (ctrl.Result, error) {
    ...
    hazelcast := &hazelcastv1.Hazelcast{}
    err := r.Get(ctx, req.NamespacedName, hazelcast)
    ...
    found := &appsv1.Deployment{}
    err = r.Get(ctx, types.NamespacedName{Name: hazelcast.Name,
                                          Namespace: hazelcast.Namespace}, found)
    if err != nil && errors.IsNotFound(err) { dep := r.deploymentForHazelcast(hazelcast) }
    ...
}
```

Go: Create Operator

```
// controllers/hazelcast_controller.go
...
Spec: appsv1.DeploymentSpec{
    Replicas: &replicas,
    Selector: &metav1.LabelSelector{
        MatchLabels: ls,
    },
    Template: corev1.PodTemplateSpec{
        ObjectMeta: metav1.ObjectMeta{
            Labels: ls,
        },
        Spec: corev1.PodSpec{
            Containers: []corev1.Container{{
                Image: "hazelcast/hazelcast",
                Name:  "hazelcast",
            }}, }, }, }, }
```

```
// api/v1/hazelcast_types.go

type HazelcastSpec struct {
    Size int32 `json:"size,omitempty"`
}
```

Go: Build, Install, Use Operator

Common Operator Steps: build, install, and use

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
$ make install
$ make deploy IMG=<user>/hazelcast-operator
$ kubectl apply -f config/samples/hazelcast_v1_hazelcast.yaml
```

What does "Operator SDK: Go" mean to You?

- Implementation is imperative and requires more work and caution
- Go language is well integrated with Kubernetes
- No limits on the functionality
- Operator boilerplate files are automatically generated

Agenda

- Introduction ✔
- Tools for building Operators
  - Operator SDK ✔
    - Helm ✔
    - Ansible ✔
    - Go ✔
  - Operator Frameworks: KOPF, Java Operator SDK, ...
  - Bare Programming Language: Java, Kotlin, C#, ...
- Summary

Operator Frameworks

KOPF

C# Operator SDK

NodeJS
Operator Framework

Java
Operator SDK

Roperator

KOPF

Star 321

C# Operator SDK

Star 1

NodeJS
Operator Framework

Star 34

Java
Operator SDK

Star 107

Roperator

Star 132

KOPF

Star 321

C# Operator SDK

Star 1

NodeJS
Operator Framework

Star 34

Java
Operator SDK

Star 107

Roperator

Star 132

Star 962

KOPF

Star 321

C# Operator SDK

Star 1

NodeJS
Operator Framework

Star 34

Java
Operator SDK

Star 107

Roperator

Star 132

Star 962

zalando

KOPF: Create Operator

1. Implement configuration and manifest files

- `Dockerfile`
- `hazelcast.crd.yaml`
- `role.yaml, role_binding.yaml`
- `operator.yaml`
- `hazelcast.yaml`

2: Implement operator logic in operator.py

KOPF: Create Operator

```python
@kopf.on.create('hazelcast.my.domain','v1','hazelcasts')
def create_fn(spec, **kwargs):
    doc = create_deployment(spec)
    kopf.adopt(doc)

    api = pykube.HTTPClient(pykube.KubeConfig.from_env())
    deployment = pykube.Deployment(api, doc)
    deployment.create()

    api.session.close()

    return {'children': [deployment.metadata['uid']]}
```

```python
def create_deployment(spec):
    return yaml.safe_load(f"""
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: hazelcast
        spec:
          replicas: {spec.get('size', 1)}
          selector:
            matchLabels:
              app: hazelcast
          template:
            metadata:
              labels:
                app: hazelcast
            spec:
              containers:
                - name: hazelcast
                  image: "hazelcast/hazelcast"
    """)
```

KOPF: Build, Install, Use Operator

Common Operator Steps: build, install, and use

```
$ docker build -t <user>/hazelcast-operator .
$ docker push <user>/hazelcast-operator
$ kubectl apply -f hazelcast.crd.yaml
$ kubectl apply -f role.yaml
$ kubectl apply -f role_binding.yaml
$ kubectl apply -f operator.yaml
$ kubectl apply -f hazelcast.yaml
```

What does "Operator Framework" mean to You?

- Less developed and popular than Operator SDK
- No project scaffolding and boilerplate code generation
- No limits on the functionality
- Kubernetes clients are usually inferior to the Go Kubernetes client

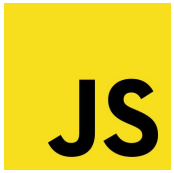
Agenda

- Introduction ✔
- Tools for building Operators
  - Operator SDK ✔
    - Helm ✔
    - Ansible ✔
    - Go ✔
  - Operator Frameworks: KOPF, Java Operator SDK, … ✔
  - Bare Programming Language: Java, Kotlin, C#, ...
- Summary

Bare Programming Language


Groovy


node
JS


JS


C#

Java + Quarkus: Create Operator

1. Implement configuration and manifest files

- `hazelcast.crd.yaml`
- `role.yaml, role_binding.yaml`
- `operator.yaml`
- `hazelcast.yaml`

2: Implement Java classes with the operator logic

Java + Quarkus: Create Operator

```
List hazelcastDeployments = client.apps().deployments().list().getItems().stream()
        .filter(ownerRefMatches)
        .collect(toList());

 if (hazelcastDeployments.isEmpty()) {
     client.apps().deployments().create(newDeployment(resource));
 } else {
     for (Deployment deployment : hazelcastDeployments) {
         setSize(deployment, resource);
         client.apps().deployments().createOrReplace(deployment);
     }
  }
```

Java + Quarkus: Create Operator

```
public class ClientProvider {

    @Produces
    @Singleton
    @Named("namespace")
    private String findNamespace() throws IOException {
        return new String(Files.readAllBytes(Paths.get("/var/run/secrets/kubernetes.io/serviceaccount/namespace")));
    }

    @Produces
    @Singleton
    KubernetesClient newClient(@Named("namespace") String namespace) {
        return new DefaultKubernetesClient().inNamespace(namespace);
    }

    @Produces
    @Singleton
    NonNamespaceOperation<HazelcastResource, HazelcastResourceList, HazelcastResourceDoneable, Resource<HazelcastResource, HazelcastResourceDoneable>> makeCustomResourceClient(
            KubernetesClient defaultClient, @Named("namespace") String namespace) {

        KubernetesDeserializer.registerCustomKind("hazelcast.my.domain/v1", "Hazelcast", HazelcastResource.class);

        CustomResourceDefinition crd = defaultClient
                .customResourceDefinitions()
                .list()
                .getItems()
                .stream()
                .filter(d -> "hazelcasts.hazelcast.my.domain".equals(d.getMetadata().getName()))
                .findAny()
                .orElseThrow(
                        () -> new RuntimeException(
                                "Deployment error: Custom resource definition \"hazelcasts.hazelcast.my.domain\" not found."));

        return defaultClient
                .customResources(crd, HazelcastResource.class, HazelcastResourceList.class, HazelcastResourceDoneable.class)
                .inNamespace(namespace);
    }
    }
```

Java + Quarkus: Create Operator

```java
@ApplicationScoped
public class DeploymentInstaller {

    @Inject
    private KubernetesClient client;

    @Inject
    private HazelcastResourceCache cache;

    void onStartup(@Observes StartupEvent _ev) {
        new Thread(this::runWatch).start();
    }

    private void runWatch() {
        cache.listThenWatch(this::handleEvent);
    }

    private void handleEvent(Watcher.Action action, String uid) {
        try {
            HazelcastResource resource = cache.get(uid);
            if (resource == null) {
                return;
            }

            Predicate ownerRefMatches = deployments -> deployments.getMetadata().getOwnerReferences().stream()
                    .anyMatch(ownerReference -> ownerReference.getUid().equals(uid));


            List hazelcastDeployments = client.apps().deployments().list().getItems().stream()
                    .filter(ownerRefMatches)
                    .collect(toList());

            if (hazelcastDeployments.isEmpty()) {
                client.apps().deployments().create(newDeployment(resource));
            } else {
                for (Deployment deployment : hazelcastDeployments) {
                    setSize(deployment, resource);
                    client.apps().deployments().createOrReplace(deployment);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }

    private Deployment newDeployment(HazelcastResource resource) {
        Deployment deployment = client.apps().deployments().load(getClass().getResourceAsStream("/deployment.yaml")).get();
        setSize(deployment, resource);
        deployment.getMetadata().getOwnerReferences().get(0).setUid(resource.getMetadata().getUid());
        deployment.getMetadata().getOwnerReferences().get(0).setName(resource.getMetadata().getName());
        return deployment;
    }

    private void setSize(Deployment deployment, HazelcastResource resource) {
        deployment.getSpec().setReplicas(resource.getSpec().getSize());
    }
     }
```

Java + Quarkus: Create Operator

```java
@ApplicationScoped
public class DeploymentInstaller {

    @Inject
    private KubernetesClient client;

    @Inject
    private HazelcastResourceCache cache;

    void onStartup(@Observes StartupEvent _ev) {
        new Thread(this::runWatch).start();
    }

    private void runWatch() {
        cache.listThenWatch(this::handleEvent);
    }

    private void handleEvent(Watcher.Action action, String uid) {
        try {
            HazelcastResource resource = cache.get(uid);
            if (resource == null) {
                return;
            }

            Predicate ownerRefMatches = deployments -> deployments.getMetadata().getOwnerReferences().stream()
                    .anyMatch(ownerReference -> ownerReference.getUid().equals(uid));


            List hazelcastDeployments = client.apps().deployments().list().getItems().stream()
                    .filter(ownerRefMatches)
                    .collect(toList());

            if (hazelcastDeployments.isEmpty()) {
                client.apps().deployments().create(newDeployment(resource));
            } else {
                for (Deployment deployment : hazelcastDeployments) {
                    setSize(deployment, resource);
                    client.apps().deployments().createOrReplace(deployment);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }

    private Deployment newDeployment(HazelcastResource resource) {
        Deployment deployment = client.apps().deployments().load(getClass().getResourceAsStream("/deployment.yaml")).get();
        setSize(deployment, resource);
        deployment.getMetadata().getOwnerReferences().get(0).setUid(resource.getMetadata().getUid());
        deployment.getMetadata().getOwnerReferences().get(0).setName(resource.getMetadata().getName());
        return deployment;
    }

    private void setSize(Deployment deployment, HazelcastResource resource) {
        deployment.getSpec().setReplicas(resource.getSpec().getSize());
    }
     }
```

```java
@ApplicationScoped
public class HazelcastResourceCache {

    private final Map<String, HazelcastResource> cache = new ConcurrentHashMap<>();

    @Inject
    private NonNamespaceOperation<HazelcastResource, HazelcastResourceList, HazelcastResourceDoneable, Resource<HazelcastResource, HazelcastResourceDoneable>> crClient;

    private Executor executor = Executors.newSingleThreadExecutor();

    public HazelcastResource get(String uid) {
        return cache.get(uid);
    }

    public void listThenWatch(BiConsumer<Watcher.Action, String> callback) {

        try {
            // list
            crClient
                    .list()
                    .getItems()
                    .forEach(resource -> {
                                cache.put(resource.getMetadata().getUid(), resource);
                                String uid = resource.getMetadata().getUid();
                                executor.execute(() -> callback.accept(Watcher.Action.ADDED, uid));
                            }
                    );

            // watch
            crClient.watch(new Watcher() {
                @Override
                public void eventReceived(Action action, HazelcastResource resource) {
                    try {
                        String uid = resource.getMetadata().getUid();
                        if (cache.containsKey(uid)) {
                            int knownResourceVersion = Integer.parseInt(cache.get(uid).getMetadata().getResourceVersion());
                            int receivedResourceVersion = Integer.parseInt(resource.getMetadata().getResourceVersion());
                            if (knownResourceVersion > receivedResourceVersion) {
                                return;
                            }
                        }
                        System.out.println("received " + action + " for resource " + resource);
                        if (action == Action.ADDED || action == Action.MODIFIED) {
                            cache.put(uid, resource);
                        } else if (action == Action.DELETED) {
                            cache.remove(uid);
                        } else {
                            System.err.println("Received unexpected " + action + " event for " + resource);
                            System.exit(-1);
                        }
                        executor.execute(() -> callback.accept(action, uid));
                    } catch (Exception e) {
                        e.printStackTrace();
                        System.exit(-1);
                    }
                }

                @Override
                public void onClose(KubernetesClientException cause) {
                    cause.printStackTrace();
                    System.exit(-1);
                }
            });
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
      }
```

Java + Quarkus: Create Operator

```
public class HazelcastResource extends CustomResource {

    private HazelcastResourceSpec spec;

    public HazelcastResourceSpec getSpec() {
        return spec;
    }

    public void setSpec(HazelcastResourceSpec spec) {
        this.spec = spec;
    }

    @Override
    public String toString() {
        String name = getMetadata() != null ?
getMetadata().getName() : "unknown";
        String version = getMetadata() != null ?
getMetadata().getResourceVersion() : "unknown";
        return "name=" + name + " version=" + version + "
value=" + spec;
    }
   }
```

Java + Quarkus: Create Operator

```
public class HazelcastResource extends CustomResource {

    private HazelcastResourceSpec spec;

    public HazelcastResourceSpec getSpec() {
        return spec;
    }

    public void setSpec(HazelcastResourceSpec spec) {
        this.spec = spec;
    }

    @Override
    public String toString() {
        String name = getMetadata() != null ?
getMetadata().getName() : "unknown";
        String version = getMetadata() != null ?
getMetadata().getResourceVersion() : "unknown";
        return "name=" + name + " version=" + version + "
value=" + spec;
    }
   }
```

```
 public class HazelcastResourceList extends
 CustomResourceList<HazelcastResource> {
     // empty
    }
```

Java + Quarkus: Create Operator

```
public class HazelcastResource extends CustomResource {

    private HazelcastResourceSpec spec;

    public HazelcastResourceSpec getSpec() {
        return spec;
    }

    public void setSpec(HazelcastResourceSpec spec) {
        this.spec = spec;
    }

    @Override
    public String toString() {
        String name = getMetadata() != null ?
getMetadata().getName() : "unknown";
        String version = getMetadata() != null ?
getMetadata().getResourceVersion() : "unknown";
        return "name=" + name + " version=" + version + "
value=" + spec;
    }
   }
```

```
 public class HazelcastResourceList extends
 CustomResourceList<HazelcastResource> {
     // empty
    }
```

```
@JsonDeserialize
@RegisterForReflection
public class HazelcastResourceSpec {

    @JsonProperty("size")
    private Integer size;

    public Integer getSize() {
        return size;
    }

    @Override
    public String toString() {
        return "size=" + size;
    }
  }
```

Java + Quarkus: Create Operator

```java
public class HazelcastResource extends CustomResource {

    private HazelcastResourceSpec spec;

    public HazelcastResourceSpec getSpec() {
        return spec;
    }

    public void setSpec(HazelcastResourceSpec spec) {
        this.spec = spec;
    }

    @Override
    public String toString() {
        String name = getMetadata() != null ?
getMetadata().getName() : "unknown";
        String version = getMetadata() != null ?
getMetadata().getResourceVersion() : "unknown";
        return "name=" + name + " version=" + version + "
value=" + spec;
    }
   }
```

```java
 public class HazelcastResourceList extends
 CustomResourceList<HazelcastResource> {
     // empty
    }
```

```java
@JsonDeserialize
@RegisterForReflection
public class HazelcastResourceSpec {

    @JsonProperty("size")
    private Integer size;

    public Integer getSize() {
        return size;
    }

    @Override
    public String toString() {
        return "size=" + size;
    }
   }
```

```java
public class HazelcastResourceDoneable extends
CustomResourceDoneable<HazelcastResource> {

    public HazelcastResourceDoneable(HazelcastResource resource,
Function<HazelcastResource, HazelcastResource> function) {
        super(resource, function);
    }
  }
```

Java + Quarkus: Build, Install, Use Operator

Build Docker image

```
$ mvn package && docker build -t <user>/hazelcast-operator .
```

Build native Docker image

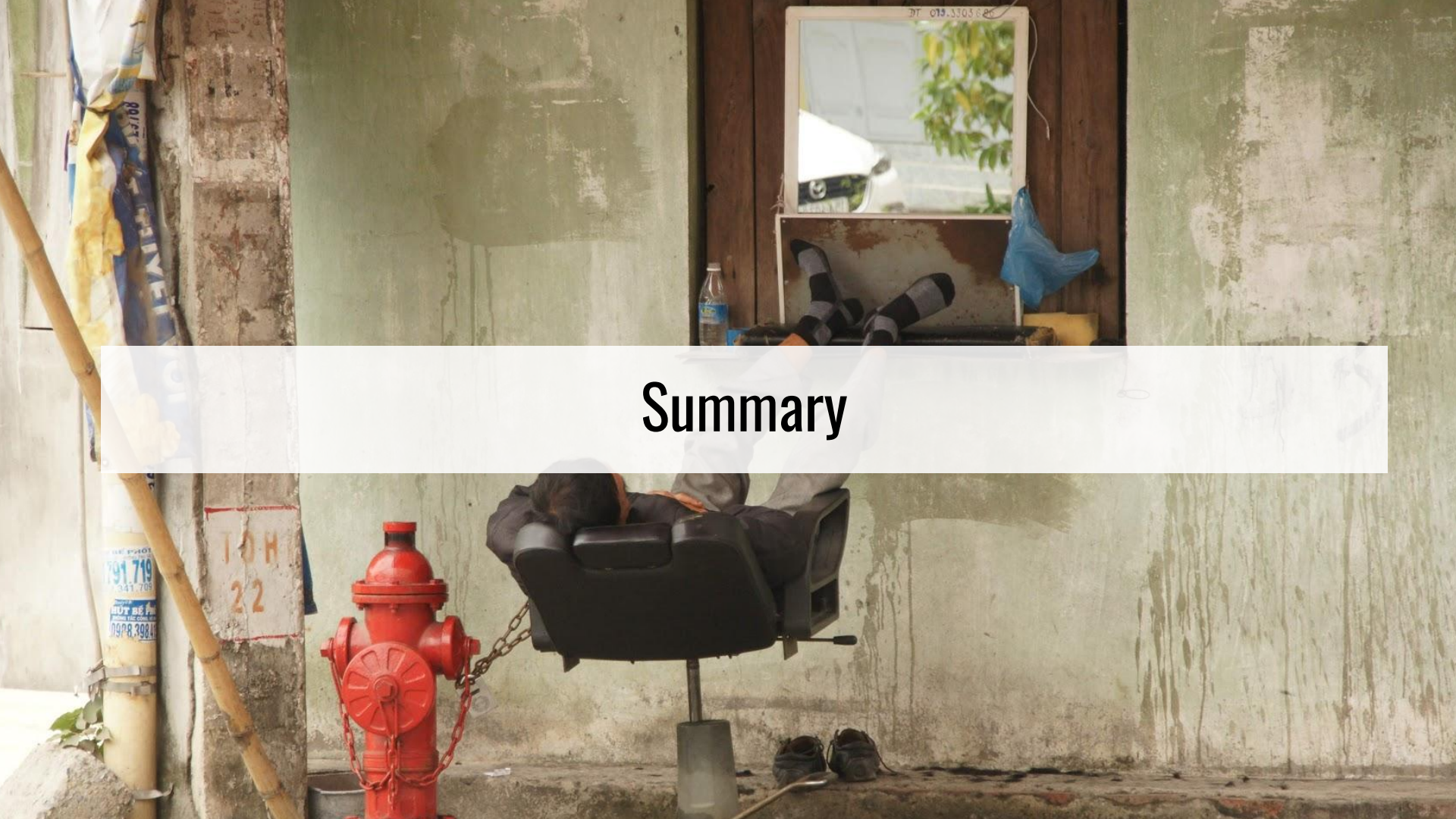
```
$ mvn package -Pnative -DskipTests -Dnative-image.docker-build=true
$ docker build -t <user>/hazelcast-operator .
```

Common Operator Steps: push, install, and use

```
$ docker push <user>/hazelcast-operator
$ kubectl apply -f hazelcast.crd.yaml
$ kubectl apply -f role.yaml
$ kubectl apply -f role_binding.yaml
$ kubectl apply -f operator.yaml
$ kubectl apply -f hazelcast.yaml
```

What does "Bare Programming Language" mean to You?

- Always means writing more code
- Check if your language has a good Kubernetes client library
- No limits on the functionality
- Only reason: single programming language in your organization



Summary

Which tool should I use for my operator?


Prometheus


Prometheus


GO


Prometheus


GO


Cockroach DB


Prometheus


GO


Cockroach DB


HELM


Prometheus


GO


Cockroach DB


HELM


DATADOG


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk


HELM


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk


HELM


Jenkins


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk


HELM


Jenkins


GO


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk


HELM


Jenkins


GO


redis


Prometheus


GO


Cockroach DB


HELM


DATADOG


GO


snyk

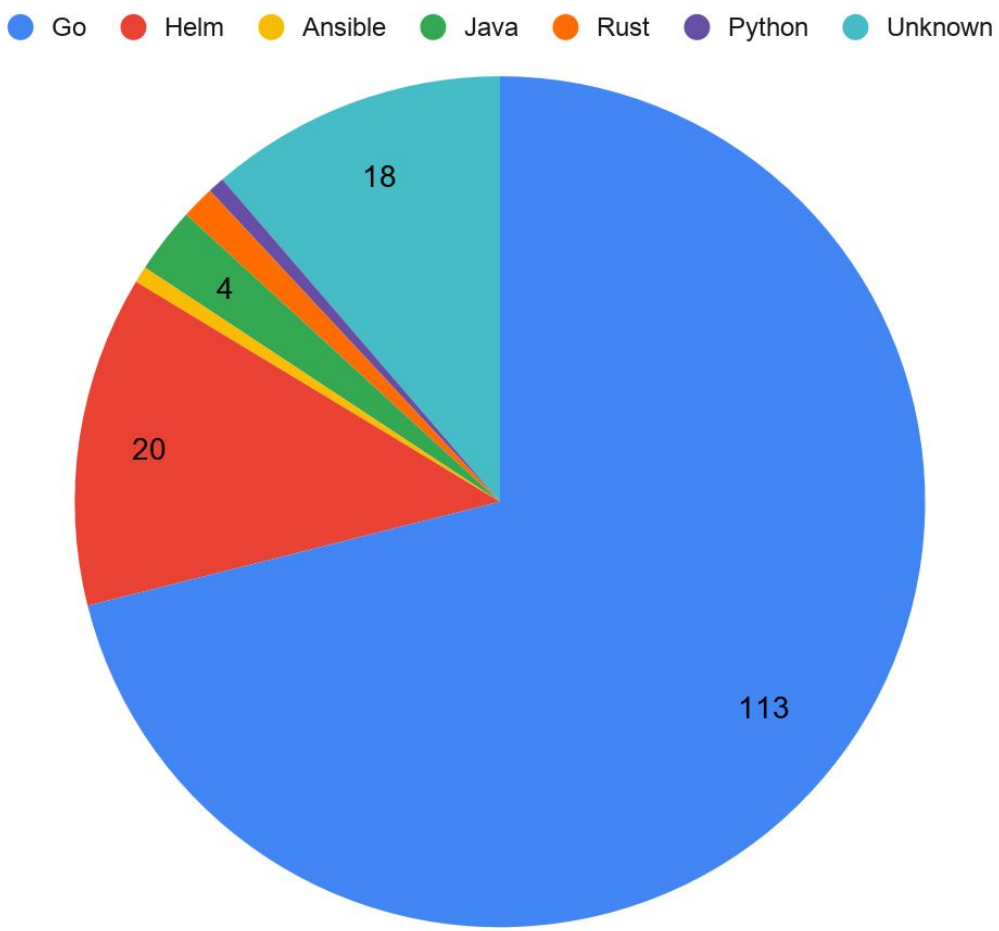

HELM


Jenkins


GO


redis


GO


Go
Helm
Ansible
Java
Rust
Python
Unknown
18
4
20
113

Hint 1: If you already have a Helm chart for your software and you don’t need any complex capability levels

Hint 1: If you already have a Helm chart for your software and you don't need any complex capability levels

Operator SDK: Helm

Hint 2: If you want to create your operator quickly and you don’t need any complex capability levels

Hint 2: If you want to create your operator quickly and you don't need any complex capability levels

Operator SDK: Helm

Hint 3: If you want complex features or/and be flexible about any future implementations

Hint 3: If you want complex features or/and be flexible about any future implementations

Operator SDK: Go

Hint 4: If you want to keep a single programming language in your organization

Hint 4: If you want to keep a single programming language in your organization

- If a popular Operator Framework exists for your language or/and you want to contribute to it

Hint 4: If you want to keep a single programming language in your organization

- If a popular Operator Framework exists for your language or/and you want to contribute to it

Operator Framework

Hint 4: If you want to keep a single programming language in your organization

- If a popular Operator Framework exists for your language or/and you want to contribute to it

Operator Framework

- If no popular Operator Framework exists for your programming language

Hint 4: If you want to keep a single programming language in your organization

- If a popular Operator Framework exists for your language or/and you want to contribute to it

Operator Framework

- If no popular Operator Framework exists for your programming language

Bare Programming Language

Hint 5: If none of the mentioned

Hint 5: If none of the mentioned

Operator SDK: Go

Resources

- Code for this presentation: https://github.com/leszko/build-your-operator
- Operator SDK: https://sdk.operatorframework.io/
- Java + Quarkus operator description: https://www.instana.com/blog/writing-a-kubernetes-operator-in-java-part-1/
- KOPF (Kubernetes Operators Framework): https://kopf.readthedocs.io/en/stable/

Thank You!

Rafal Leszko

@RafalLeszko
rafalleszko.com

Q & A

Rafal Leszko

@RafalLeszko
rafalleszko.com