Understanding the VCA

Juju is a universal operator lifecycle manager
“How can we manage all these things?”

Configuration Management: Terraform, Ansible

Declarative Deployment: Helm

Imperative Lifecycle Management: Operators
A ‘Kubernetes operator’ is a container which drives other containers.
Operators handle container lifecycle complexity

- ✓ install
- ✓ configure
- ✓ upgrade
- ✓ remove

“Automate all the detail of running this application on K8s”

An operator is ops code.

A charm is an operator package.
App domain knowledge, distilled into code

Application code is open source.
Why not share the operations code too?
“I found nine operators of Cassandra but none of them seem very good and I don’t know which one to use”
Writing great operators is hard

- UX / CLI
- Configuration
- Component permutations
- Pod-based operator limitations
- Distributed systems
“How can we improve operators?”

Configuration Management
- Terraform, Ansible

Declarative Deployment
- Helm

Imperative Lifecycle Management

Operators

Model-driven Operators
- Juju
- VCA
A better way to build and use operators

✓ Model-driven operator lifecycle manager (OLM)
✓ Composition and integration of operators
✓ Standardised UX / CLI and configuration
✓ Parameterised async Day 2 operations
✓ Fine-grained sidecar workload control
✓ Python operator framework with Golang OLM
✓ Much, much less YAML, much simpler operator code
But first, a demo!
Kubeflow
Different scenarios

Edge Example

Data Scientist Example

Enterprise Training Example
Much simpler YAML
bundle: kubernetes
applications:
seldon-core: { charm: seldon-core-15, scale: 1 }
argo-controller: { charm: argo-controller-14, scale: 1 }
pipelines-persistence: { charm: pipelines-persistence-13, scale: 1 }
pipelines-scheduledworkflow: { charm: pipelines-scheduledworkflow-15, scale: 1 }
tf-job-operator: { charm: tf-job-operator-13, scale: 1 }
minio: { charm: minio-15, scale: 1 }
pytorch-operator: { charm: pytorch-operator-14, scale: 1 }
pipelines-db: { charm: "cs:~charmed-osm/mariadb-k8s-34", scale: 1 }
pipelines-api: { charm: pipelines-api-14, scale: 1 }
relations:
  - [ argo-controller, minio ]
  - [ pipelines-api, pipelines-db ]
  - [ pipelines-api, pipelines-persistence ]
  - [ "pipelines-api:minio", "minio:minio" ]
series: ~
description: ~
Real time dynamic integration
Automated integration in production
Declarative integration
Composition and integration

“Do one thing very well”
Composition and integration

“provide prometheus”  “require prometheus”
name: prometheus2
summary: Monitoring system and time...

requires:
  graf:  
    interface: grafana-source

name: grafana
summary: Graph and Dashboard builder...

provides:
  graf:  
    interface: grafana-source
Composition and integration

“relate prometheus and grafana”
Small, composable operator
We model integration explicitly
Complexity
Composition

Model
Multi cloud integration
The model
Model-driven operators

✓ RBAC on model permissions
✓ Capacity and scale
✓ Network attachments
✓ Storage classes
✓ Architecture & operations
✓ Integration
✓ Placement
How do operators communicate?
Juju is an Operator Lifecycle Manager (OLM)

"Relate those two apps"
Consistent UX and CLI for all operators
juju deploy prometheus grafana
juju config prometheus foo=322 bar=isolated
juju relate prometheus grafana
juju scale prometheus 3
juju deploy kubeflow-pipelines istio
juju config kubeflow-pipelines driver=cuda angio=done
juju relate istio kubeflow-pipelines
juju scale kubeflow-pipelines 3
Consistent application operations
deploy
config
relate
scale
day 2

storage
network
permissions
compute

{ Consistent application operations

{ Consistent business execution
Fine-grained workload control by operators
Traditional K8s operator in separate pod

- On different machines
- No IPC or local comms
- No file sharing
- No separate state per unit
Fine-grained control with sidecar placement

- Always co-located on host
- Can use SHM or UNIX sockets
- Can share files with workload
- Operator scales too
- Better control, debugging
Both are supported
Golang and Python

OLM in Golang
- Highly concurrent
- Events and messaging
- Multi model multi app
- Performance critical
- Highly available

Charms in Python
- Simpler collaboration
- Integration code only
- Serialized event handling
- Control plane for one app
- Community-centric
The Juju OLM distributes events to operators
"Configure that app"
"Integrate those apps"
"Scale that app"

"Upgrade yourself"
"Integrate with Grafana"

"Prometheus is waiting"
"Scale yourself to 3 units"
Python operator framework is a clean event handling loop
class MyCharm(CharmBase):

def __init__(self, *args):
    super().__init__(*args)
    self.framework.observe(self.on.config_changed, self)

def on_config_changed(self, event):
    url = self.model.config[“url”]
    # ...
You can also charm traditional apps
Solve for both infrastructure and apps
On machines

The scripts work locally to that machine

On Kubernetes

The scripts in one container act on other containers
Model driven operations - Machine & Kubernetes
Universal operators

Kubernetes

VM / Cloud

Bare metal
Integration across generations

Kubernetes

VM / Cloud

Bare metal
The Open Operator Collection
Universal operators for Linux, Windows and Kubernetes apps

Read our manifesto
Reusable, composable operators

- Easier to write operators
- Easier to deploy operators
- Easier to share operators
- Easier to integrate operators

https://charmhub.io/
Thank you. Questions?