FERMYON

Introduction to SpinKube

And happy birthday!

June 2024 Cloud Native and Kubernetes Oslo

Me and My K8s Journey

Mikkel Mørk Hegnhøj

Head of Product and DevRel at Fermyon since 2022.

Product Management at Microsoft, focused on developer experiences and cloud services.

K8s journey began around 1.17 ~2019

Major K8s contribution – WINDOWS CONTAINERS!!!

Also worked on Virtual Kubelet

Reflections on Kubernetes
What have we learned and adopted?

How to use Cloud

Compute and workloads can (and should be) be decoupled (pets vs. cattle v2.0)

Compute resources offered through a data plane of immutable resources (computers with CPUs, RAM, Network, Storage, etc.) – An abstraction

Workloads are packaged in a standardized format, and described and reconciled through a control plane

Coordinated collaboration in the open is possible

Looking Ahead

Your infrastructure is ubiquitous

Private datacenters, public clouds, branches, SaaS, etc.

Data planes offers capabilities

Hardware, availability, jurisdiction, emission, higher-level services, etc.

Control planes orchestrates workload constraints across data planes

Your organizations control plane

What Next for Kubernetes

Will there ever be a Kubernetes 2.0?

Will there be a Kubernetes v1.100?

Control Planes -> Cross-plane, Radius

Capabilities-approach to platforms -> Backstage, WebAssembly Component Model

Better compute format (next generation) -> WebAssembly

SpinKube

An open-source project that streamlines developing, deploying and operating WebAssembly workloads in Kubernetes

Microsoft REPLY LIQUID SUSE FERMYON

The developer tool for building WebAssembly microservices and web applications

SpinKube

Hyper-efficient serverless on Kubernetes, powered by WebAssembly

SPIN

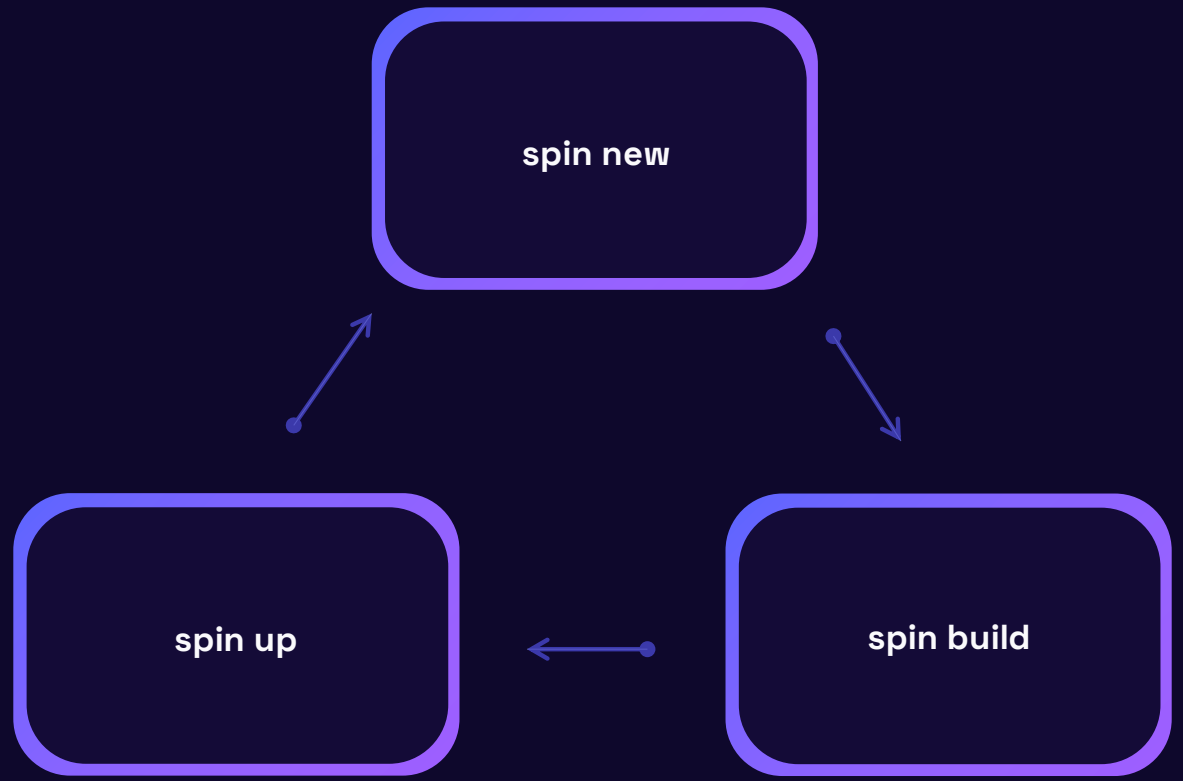
Spin is an open-source project, built with open standards like WASI and the WebAssembly Component Model.

Developer tool for building and running event-driven serverless WebAssembly applications

4.8K GITHUB ★

75+ CONTRIBUTORS

github.com/fermyon/spin



SPIN

Spin is an open-source project, built with open standards like WASI and the WebAssembly Component Model.

AT A GLANCE:

Serverless AI NEW

Quickly test and run inferencing workloads with LLMs.

Key/Value Store

Easily persist data in your apps with a built-in KV store.

Powerful CLI

Easy to create, run and deploy projects - in as little as 66 seconds.

SQLite Database

Add SQLite data to your app with an always-available SQLlite DB.

DEV EXPERIENCE:

- Supports almost any programming language
- Easy to debug with included helper commands

Demo

Why Spin WebAssembly applications?

Small packages

A hello-world Spin WebAssembly application written in Rust is a 284kB OCI Artifact.

They start up almost immediately

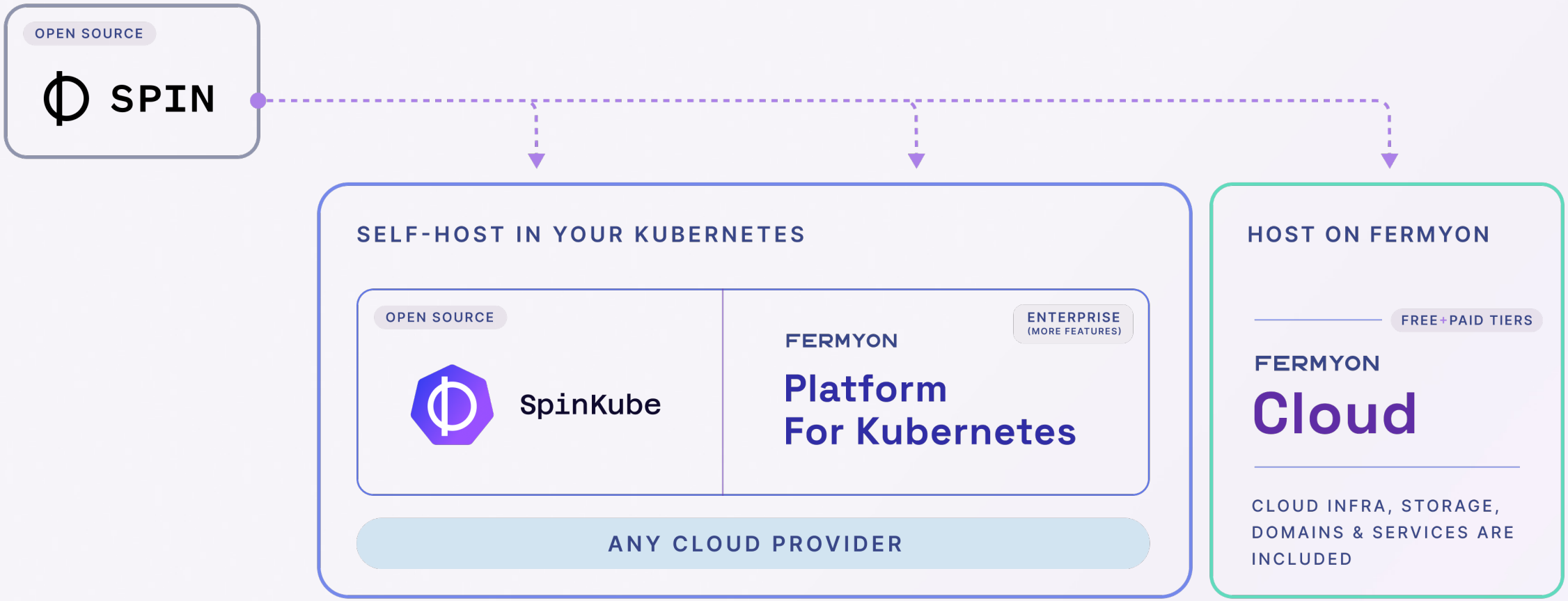
A Spin WebAssembly component will start in less than a millisecond.

They are sandboxed

WebAssembly components are sandboxed and are by default denied access to resources on the system.

They are portable across processing architectures

You can swap out the underlying nodes processing architecture without having to produce separate pipelines and deploy artifacts.


OPEN SOURCE
SPIN
SELF-HOST IN YOUR KUBERNETES
OPEN SOURCE
SpinKube
ENTERPRISE
(MORE FEATURES)
FERMYON
Platform
For Kubernetes
ANY CLOUD PROVIDER
HOST ON FERMYON
FREE+PAID TIERS
FERMYON
Cloud
CLOUD INFRA, STORAGE,
DOMAINS & SERVICES ARE
INCLUDED

SpinKube

> Utilizing Spin WebAssembly with SpinKube on Microsoft's Azure Kubernetes Service helps us to achieve faster scalability, and reach higher density without the need to dramatically change our operational posture. With that we've been able to take a Kubernetes batch process of tens of thousands of orders and cut the compute cost by 60% without trading off performance,"
>
> **— Kai Walter, Distinguished Architect, ZEISS**

SpinKube

Spin Operator

A Kubernetes operator to deploy Spin applications.

Containerd-shim-spin

A ContainerD shim for running Spin Applications.

Runtime-class manager

A simple way to install Wasm runtimes.

Spin kube plug-in

A Spin plugin for interacting with Kubernetes.

Demo

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: annotations-spinapp
spec:
  image: "ghcr.io/spinkube/hello:v0.13.0"
  replicas: 1
  executor: containerd-shim-spin
  serviceAnnotations:
    key: value
  deploymentAnnotations:
    key: value
    multiple-keys: are-supported
  podAnnotations:
    key: value
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. **Resource limits**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: resource-requirements-spinapp
spec:
  image: "ghcr.io/spinkube/hello:v0.13.0"
  replicas: 1
  executor: containerd-shim-spin
  resources:
    limits:
      cpu: 500m
      memory: 500Mi
    requests:
      cpu: 100m
      memory: 400Mi
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. **Liveness/readiness probes**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: healthchecks-spinapp
spec:
  image: "ghcr.io/spinkube/hello:v0.13.0"
  replicas: 1
  executor: containerd-shim-spin
  checks:
    liveness:
      httpGet:
        path: "/hello"
    readiness:
      httpGet:
        path: "/go-hello"
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. Liveness/readiness probes
4. **Scalers like HPA and KEDA**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: hpa-spinapp
spec:
  image: ghcr.io/spinkube/cpu-load-gen:v1
  executor: containerd-shim-spin
  enableAutoscaling: true
  resources:
    limits:
      cpu: 500m
      memory: 500Mi
    requests:
      cpu: 100m
      memory: 400Mi
```

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: spinapp-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hpa-spinapp
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. Liveness/readiness probes
4. Scalers like HPA and KEDA
5. **Private container registries**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: private-image-spinapp
spec:
  image: "ghcr.io/<username>/<image>:<tag>"
  imagePullSecrets:
    - name: spin-image-secret
  replicas: 1
  executor: containerd-shim-spin
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. Liveness/readiness probes
4. Scalers like HPA and KEDA
5. Private container registries
6. **Volume mounts**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: volume-mount-spinapp
spec:
  image: "ghcr.io/spinkube/hello:v0.13.0"
  replicas: 1
  executor: containerd-shim-spin
  volumes:
    - name: example-volume
      persistentVolumeClaim:
        claimName: example-pv-claim
  volumeMounts:
    - name: example-volume
      mountPath: "/mnt/data"
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. Liveness/readiness probes
4. Scalers like HPA and KEDA
5. Private container registries
6. Volume mounts
7. **Variables**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: variables-spinapp
spec:
  image: "ghcr.io/spin-kube/vars:v1"
  replicas: 1
  executor: containerd-shim-spin
  variables:
    - name: greetee
      value: Fermyon
```

IN PRACTICE

Extend your Spin App with Kubernetes Primitives

1. Annotations
2. Resource limits
3. Liveness/readiness probes
4. Scalers like HPA and KEDA
5. Private container registries
6. Volume mounts
7. Variables
8. **Runtime config**

```yaml
apiVersion: core.spinoperator.dev/v1alpha1
kind: SpinApp
metadata:
  name: runtime-config
spec:
  image: "ghcr.io/spinkube/hello:v0.13.0"
  replicas: 1
  executor: containerd-shim-spin
  runtimeConfig:
    sqliteDatabases:
      - name: "default"
        type: "libsql"
        options:
          - name: "url"
            value: "https://mydb.libsql.example.com"
          - name: "token"
            valueFrom:
              secretKeyRef:
                name: "my-super-secret"
                key: "turso-token"
```

Why do we need Spin WebAssembly applications in Kubernetes?

Small packages

A hello-world Spin WebAssembly application written in Rust is a 284kB OCI Artifact.

They start up almost immediately

A Spin WebAssembly component will start in less than a millisecond.

They are sandboxed

WebAssembly components are sandboxed and are by default denied access to resources on the system.

They are portable across processing architectures

You can swap out the underlying nodes processing architecture without having to produce separate pipelines and deploy artifacts.

Where can I find it?

- Containerd-shim-spin
- Docker Desktop
- Rancher Desktop
- Podman
- K3s
- MiniKube
- MicroK8s
- KinD
- Azure Kubernetes Service
- Amazon Web Services (Ubuntu only)
- Google Cloud Platform (Ubuntu only)
- Digital Ocean Kubernetes
- CIVO Kubernetes (TALOS!!!)

What Next for Kubernetes

Will there ever be a Kubernetes 2.0?

Will there be a Kubernetes v1.100?

Control Planes -> Cross-plane, Radius

Capabilities-approach to platforms -> Backstage, WebAssembly Component Model

Better compute format (next generation) -> WebAssembly

Resources

SpinKube – https://spinkube.dev
Spin – https://fermyon.com/spin