The Elements of Kubernetes

Aaron Schlesinger
Microsoft Azure Containers
Lead, SIG-Service-Catalog

Why we're here

★ Kubernetes is growing fast

★ Usage, development, *projects*

★ *We're in the wild west*

Building an app for Kubernetes

Dev/Test Containerize CI/CD

Staging Pre-prod Prod

Monitoring Tracing Logging Observability Resilience

Lots to figure out

No one size fits all

We need a “north star” for people building cloud-native apps

North Star

- ★ Help app operators/developers decide what to do
- ★ Stand up to rapid technology changes
- ★ Guide SIGs

Best practices, not rules

What we have now

- ★ Opinions
- ★ Evidence
- ★ Fragmentation

I’ve seen the good & the bad

I’m here to propose ideas

Observability is golden

Observability is golden

★ Kubernetes schedules containers

★ Kubernetes observes containers

★ You observe containers

Kubernetes observing your app

★ Resource limits
★ Readiness & liveness checks
★ HPA

You observing your app

Adopt a rich ecosystem of cloud native tools:

★ Logging
★ Service mesh
★ Tracing

When all else fails, crash

Crash-only software

★ You’ll have bugs, network outages, disk issues, etc...

★ Kubernetes *is* your retry loop

What that might look like

★ Your app connects to a DB on startup
★ App fails to connect, crashes
★ Tracing sees the conn. failure
★ Kubernetes restarts it
★ Monitoring, log aggregation pick up the restart
★ Alerting notifies if too many restarts
★ ...

Unordered is better than ordered

Unordered is better than ordered

- ★ Kubernetes & your app are distributed systems
- ★ **Ordering is very hard in distributed systems**
- ★ Try not to rely on ordering

But sometimes you need it

Use Kubernetes primitives.

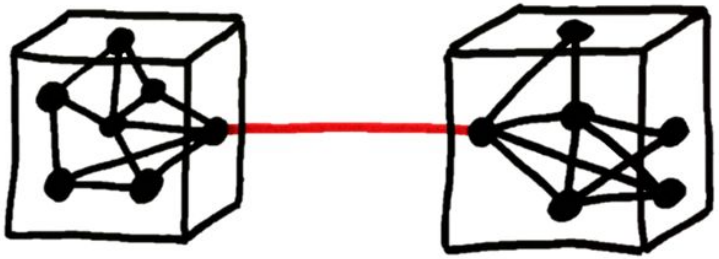
- ★ Sidecar for locks, leader election
- ★ Resource versions in Kubernetes resources
- ★ Init containers (can be messy)

Loose coupling is better than tight coupling

Loose coupling is better than tight

★ Kubernetes is always watching

★ Your app should tolerate dynamism



What that might look like

- ★ Pod => Pod messaging via `Service`s
- ★ Crash if you can’t connect (crash-only)
- ★ Look for Kube resources via labels, not names

… But tight coupling isn’t *always* wrong

Tight coupling isn’t *always* wrong

★ Pods have >1 containers on purpose

★ Run tightly coupled containers in a pod

What that might look like

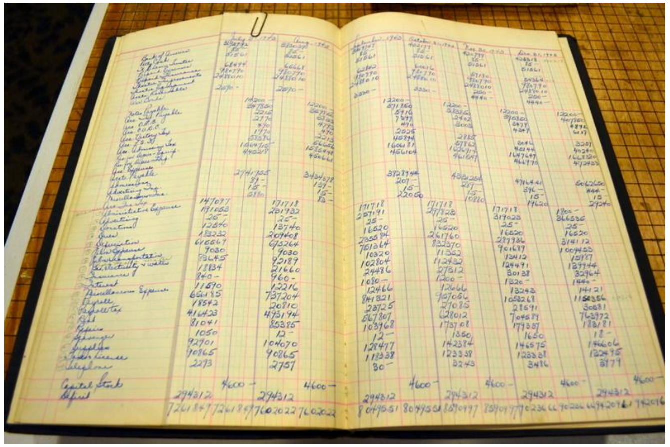
★ Envoy

★ Fluentd logging driver

★ Metrics

Record your configuration

Record your configuration



- ★ Kubernetes APIs are declarative
- ★ Keep the latest working configuration in your repository
- ★ Let Kubernetes reconcile

What might that look like

Ask for the least

Ask for the least

If you're:

★ Configuring RBAC permissions
★ Configuring containers in a pod
★ Asking for CPU shares or memory
★ Asking for disk space with a PersistentVolumeClaim

Ask for the fewest possible of them. Leave more for Kubernetes

What that might look like

★ Read-only permissions for your monitoring system
★ One CPU share for each web frontend
★ Minimal disk for your log aggregator
★ Tiny memory for your local proxy

Build on the shoulders of giants

Build on the shoulders of giants

- ★ Kubernetes provides a big API
  - ○ … to abstract functionality that's *hard to get right*
- ★ Maybe the community doesn't do what you need
  - ○ … but try to find the next best thing and build atop

What that might look like

Helm for managing your app lifecycle

★ Or the Helm API (which is gRPC)

Traefik for Ingress

★ Or Traefik => service mesh => your app

Fluentd for Logging

★ Or app => local translator => fluentd

Parting thoughts: where should we go from here?

Who should define our guidelines?

★ I’ve started the conversation here

★ We all have a wealth of experience

★ We need to share it

We need your thoughts

https://github.com/arschles/kube-best-practices

Thank you

aaron.schlesinger@microsoft.com
@arschles
github.com/arschles

Community Extras

Pay it forward

Pay it forward

- ★ If you “build on the shoulders of giants,” open source it
- ★ You’ll be moving cloud native forward
- ★ We’ll progress as community & concept because of your work

Disagree constructively

Disagree constructively

★ Cloud native is young

★ If you disagree with a choice, others do too

★ Make your voice heard, and **offer solutions**