The Elements of Kubernetes

Aaron Schlesinger
O’Reilly Software Architecture Conference
Feb 5, 2019
Welcome!

- Cloud Advocate, Microsoft Azure
- Emeritus Chair, Kubernetes SIG-Service-Catalog
- Kubernetes contributor
Why We’re Here

• Kubernetes is growing **fast**
• Cloud Native is brand new
• *We’re in the wild west*
Fundamental Shift: Cloud Native

- Dev/Test
- Containerize
- CI/CD
- Staging
- Pre-prod
- Prod
- Monitoring
- Tracing
- Logging
- Observability
- Resilience
We have lots to figure out.

But one size doesn’t fit all.
What We Have Now

• Opinions
• Evidence
• Fragmentation
We need a “north star” for people building cloud-native apps.
North Star?

- Help app operators decide what technology to use
- Help developers think about their workflows
- Evolve with technology changes

Best practices, not rules
I’ve seen the good & the bad.
I’m here to propose ideas.
Kubernetes (k8s) Terminology in Pictures

Pod

Container

Container

Container

Service
Observability is golden.
Observability is Golden

- Kubernetes schedules & observes containers
- But your team does too!
- Help both parties do their job
Kubernetes Observes Your App

• Resource limits (CPU, Memory, …)

• Custom health probes (AKA readiness / liveness checks)

• Horizontal Pod Autoscaling
You Observe Your App

• Logging
• Service Mesh
• Tracing
When all else fails, crash.
Crash-Only Software

- Realities: bugs, network outages, flaky disk, etc...
- Prior art: Erlang supervisors
- Kubernetes *is* your supervisor
Don’t Do This

```java
conn = null
while(!conn) {
    conn = connect_to_db()
}
```
Do This

conn = connect_to_db()
if !conn {
    exit(1) // tell Kube we failed!
}
Unordered is better than ordered.
Unordered is better than ordered

- Your app is a distributed system
- **Ordering is hard**
- *Loose coupling + crash only* instead
But Sometimes You Need It…

- **Pod** - all containers or none, scheduled together
- *Sidecars* for locks & leader election
- Resource versions for MVCC
- Init containers for setup
Loose coupling is better than tight coupling.
Loose Coupling is Better Than Tight Coupling

• Kubernetes is always watching

• Your app should tolerate dynamism
What That Might Look Like

- Use message passing
- Service discovery via Service abstraction
- Crash if you can’t connect (crash-only)
… But tight coupling isn’t always wrong.
Tight coupling isn’t *always* wrong

- **Pods** have 1+ containers on purpose
- Share localhost, filesystem, etc…
- As good as it gets for coupling
What That Might Look Like

Sidecars for:

- Service Mesh
- Logging
- Metrics aggregation
Record your configuration.
Record your configuration

• Kubernetes APIs are *declarative*
• Latest working config lives next to code
• Use SCM to version configs
Declarative “Manifests”

apiVersion: v1
kind: Pod
spec:
  containers:
    image: docker.io/gomods/proxy:v0.1.0
    imagePullPolicy: Always
    name: CatVideos!!!
What That Might Look Like

- Bundle your Kube templates in a Helm chart
- Helm for one-click deployment
- Helm for lifecycle management
Ask for the least.
Ask For The Least

- RBAC permissions
- Containers in a pod
- CPU shares or memory
- Disk space

*Just like the principle of least privilege.*
What That Might Look Like

• Read-only filesystem for monitoring systems
• One CPU share for web frontends
• Minimal disk for log aggregators
• Tiny memory for local proxies
Where should we go from here?
Who should define our guidelines?

- I’ve started the conversation here
- We all have a wealth of experience
- From many viewpoints
- Let’s share all of it
Contributions Welcome

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

@ arschles
github.com/ arschles
 arschles.com